BRÜCKMANN · ENGLISCH
GERITS · STEIGERS

CPC 664/6128 INTERN

EIN DATA BECKER BUCH

BRÜCKMANN · ENGLISCH
GERITS · STEIGERS

CPC 664/6128 INTERN

EIN DATA BECKER BUCH

ISBN 3-89011-135-1

INHALTSVERZEICHNIS

Einleitung

Es ist schon ungewöhnlich - das Verhalten der Firmen AMSTRAD und SCHNEIDER. Kaum hatte sich der CPC 464 Dank niedrigem Preis und ausgezeichneter Leistung auf dem heiß umkämpften Computermarkt etabliert, kam mit dem CPC 664 schon der nächste Rechner auf den Markt. Und nicht einmal 3 Monate später erschien der CPC 6128 als dritter Rechner der CPC-Reihe in den Computerläden. Auch die beiden Nachfolger des 464 fallen durch ein hervorragendes Preis/Leistungsverhältnis auf.
Mehr noch als beim 464 besticht die Vollständigkeit des Systems. Kein Streit um Dallas oder Sportschau trüben dank mitgeliefertem Farb- oder Grünmonitor das Familienleben, die lästigen und immer nur als Fußangeln nützlichen Verbindungskabel gehören der Vergangenheit an, die eingebaute Floppy verringert wesentlich den sonst üblichen Kabelsalat und kostspielige Speichererweiterungen oder Interface-Karten kann man getrost vergessen. Es ist einfach alles da, um sofort loszulegen.

Und wie man loslegen kann. Das LOCOMOTIVE-BASIC gehört unbestreitbar zum besten, was man für Geld und gute Worte bekommen kann. Besonderer Knackpunkt ist die sehr flexible und vielseitig einsetzbare Programmierung von Interrupts, die dieses BASIC parat hat.
Die excellente Grafik und die Möglichkeit der Darstellung von 80 Zeichen auf dem Bildschirm ohne zusätzliche Module und Unkosten ist bislang unübertroffen. Andere Rechner in dieser Preisklasse haben oft schon immense Probleme, 40 Zeichen pro Zeile lesbar und flimmerfrei auf den Bildschirm zu bekommen.
Die Grafik-Auflösung von 640 x 200 Punkten ist in dieser Preisklasse genau so einmalig. Vergleichbare Leistungen bietet z.B. der IBM-PC, das allerdings nur beim mindestens fünf- bis achtfachen des Preises eines CPC.
Auch die Soundmöglichkeiten des CPC sind beeindruckend. Zwar ist die Erzeugung eines Stradivari-Klangs selbst bei

geschicktester Programmierung nicht erzielbar, aber Sie haben sich ja für einen leistungsfähigen Computer und nicht für eine Geige entschieden.

Was die Geschwindigkeit angeht, so muß sich der CPC nicht verstecken. Der eingebaute Z80-Prozessor wird mit einer Taktfrequenz von 4 MHz betrieben und hat einen sehr mächtigen Befehlsvorrat. Dieser Befehlsvorrat wurde von den Entwicklern stark 'ausgereizt'. Das Ergebnis ist ein wirklich flotter BASIC-Interpreter, des seinesgleichen sucht.

Aber über kurz oder lang (sicher eher kurz) kommt bei fast allen Computerbesitzern der Wunsch nach mehr Information, mehr Wissen über den Computer, den man besitzt. Das wirklich lobenswert gute Bedienungs-Handbuch zum CPC allein reicht nicht aus. Besonders gilt dies dann, nachdem das BASIC etwas an Reiz verloren hat, und es gilt, die durch das BASIC gesteckten Grenzen in Richtung Maschinensprache zu überschreiten. Dann werden Informationen nötig, die weit über das hinausgehen, was ein Bedienungshandbuch geben kann.

Das bisher von den Büchern der INTERN-Reihe gewohnte ROM-Listing finden Sie hier in einer neuen, kompakteren Form. Zugunsten ausführlicherer Kommentare haben wir auf das eigentliche Listing verzichtet. Mit Hilfe des in diesem Buch abgedruckten Disassemblers können Sie sich jedoch jederzeit Ihr eigenes Listing erstellen. In Zukunft wird diese Art, Betriebssysteme zu dokumentieren, ohnedies angebracht sein, da die Betriebssysteme immer größere Dimensionen erreichen, deren Dokumentation einen normalen Buchumfang sprengen würde.

Ihre Autoren

1.1 Das sollten Sie von Ihrem Gerät wissen

Insgesamt enthält Ihr CPC sechs wesentliche hochintegrierte ICs. Der wichtigste Baustein eines jeden Computers, der Prozessor, ist im CPC ein Z80. Außerdem sind ein Video-Controller HD 6845, ein Parallelport 8255, ein Sound Chip AY-3-8912, der Floppy-Controller 765 und ein speziell für den CPC entwickeltes sogenanntes Gate Array eingebaut.

Der Video-Controller hat die Aufgabe, alle für den Betrieb des Monitors benötigten Signale zur Verfügung zu stellen. Auch adressiert er das Bildschirm-RAM, den Speicherbereich, in dem die darzustellenden Zeichen und die Grafik abgelegt werden. Zusätzlich erzeugt er den für die RAMs nötigen Refresh, ohne den diese ihre Information schnell verlieren würden.

Die Aufgabe des Sound Chip geht schon aus seinem Namen hervor. Die von den Konstrukteuren getroffene Wahl ist sehr gut. Der AY-3-8912 ist in vielen Computern eingesetzt worden, weil er vielseitige und weitreichende Beeinflussungsmöglichkeiten des Sounds bietet.

Der 8255 ist das 'Arbeitstier' im CPC. Seine Aufgaben sind sehr vielfältig. Das beginnt bei der Kontrolle der Tastatur, geht über die Ansteuerung des Sound Chip weiter zur Steuerung des Recorders, bestimmt verschiedene Möglichkeiten des CPC u.s.w.

In Verbindung mit einigen TTL-ICs und einem sogenannten Datenseparator bildet der Floppy-Controller 765 die Schnittstelle zu den maximal zwei Floppylaufwerken. Der Floppy-Controller übernimmt dabei fast die vollständige Steuerung der Laufwerke und erleichtert durch seine hohe 'Intelligenz' die Programmierung.

Besonders interessant ist das Gate Array. Dieser Baustein steuert so viele Dinge im CPC, daß man ihm fast den Rang eines Hilfsprozessors zusprechen könnte. So werden viele den

Bildschirm betreffende Aufgaben von ihm übernommen. Dazu gehört unter anderem die Darstellung der verschiedenen Farben und die unterschiedlichen Zeilenformate. Weiter werden alle benötigten Taktsignale im Gate Array erzeugt. Der Interrupt, der 300 mal in der Sekunde den normalen Programmablauf unterbricht, wird, wie auch die Signale für die Ansteuerung des RAM im CPC, vom Gate Array generiert.

1.1.1 Die Speicheraufteilung

Noch vor 5 Jahren galten Computer mit 16 K RAM als recht gut bestückt. Spätestens seit dem Erscheinen des C64 von Commodore haben sich die Speichergrenzen deutlich verschoben. Ein Computerhersteller kann sich nur dann ausreichend Marktchancen ausrechnen, wenn auf seinem Gerät wenigstens die 'magische 64' erscheint. Da aber die Preise für die Speicherbausteine in den letzten Monaten drastisch gesunken sind, spielt es für den Preis eines Rechners eine nur geringfügige Rolle, ob er nun mit 64 KByte (wie im 664) oder 128 KByte (6128) ausgerüstet ist.
Nun ist es nicht besonders schwer, 64 K Speicher in einem Computer unterzubringen, da die gebräuchlichen 8-Bit-Prozessoren diesen Speicherbereich alle adressieren können. Auch der Z80 des CPC kann 64 K Speicher ohne Tricks adressieren. Da jedoch im CPC nicht nur RAM, sondern auch ROM adressiert werden muß, entstehen einige Schwierigkeiten. Diese Schwierigkeiten sind im CPC sehr elegant umgangen worden. Minimale Hardware und sehr ausgefeilte Software erledigen das im CPC eingesetzte Bank-Switching, mit dem RAM und ROM bei Bedarf umgeschaltet werden. Auch die Ansteuerung der zusätzlichen 64 Kbyte RAM im 6128 ist mit Hilfe des Bank-Switching gelöst.

Im CPC 6128 ergibt sich nun das folgende Bild (Anm.: im 664 sind die beschriebenen Verhältnisse ähnlich, nur fehlt die zusätzliche 64K-RAM-Bank). Durchgängig adressiert werden 64 K RAM. 'Parallel' dazu, jedoch nicht ohne weiteres ansprechbar liegt die zusätzliche 64 KByte-Bank sowie in den

unteren 16 K (&0000 bis &3FFF) eine Hälfte des 32K-ROM. Die zweite Hälfte dieses ROMs liegt in den oberen 16 K (&C000 bis &FFFF).
Die unteren 16 K ROM beinhalten im wesentlichen das Betriebssystem des Rechners inclusiver der Routinen für die Ansteuerung des Recorders und der Speicherumschaltung. Im Betriebssystem finden sich alle Routinen, die der CPC benötigt, um z.B. ein Zeichen von der Tastatur zu lesen, ein Zeichen oder einen Grafik-Punkt auf den Bildschirm zu bringen, aber auch der Recorder und die Drucker-Schnittstelle sowie der Sound wird über das Betriebssystem bedient.
In den oberen 16 K befindet sich der BASIC-Interpreter. Zu diesem Bereich parallel liegt das ROM des AMSDOS, das alle Routinen zum Betrieb der Floppy enthält. Damit aber nicht genug, in diesen Bereich können bis zu 251 weitere ROMs geschaltet werden. In diesen ROMs können z.B. andere Programmiersprachen, BASIC-Erweiterungen oder auch Spiele untergebracht werden.

Grafisch kann man die Speicheraufteilung wie im Bild 1.1.1.1 darstellen.

1.1.2 Befehlserweiterung via RST

Um den Zugriff auf die verschiedenen ROMs möglichst zu gestalten, haben sich die Programmierer des Betriebssystems einen schönen Trick einfallen lassen. Durch spezielle Programme und der geschickten Ausnutzung der RESTART-Befehle des Z80 ergibt sich für die Restarts RST1 bis RST5 quasi eine Erweiterung des Befehlssatzes des Z80. Diese RSTs lassen sich wie übliche JMPs oder CALLs einsetzen. Bei einigen RSTs wird allerdings eine 3-Byte-Adresse verlangt. Im zusätzlichen dritten Byte wird dann bestimmt, in welches ROM der JMP oder CALL gehen soll.

	RAM BANK Ø	RAM BANK 1	ROM	ROM	
FFFF					
	Block 3	Block 3	BASIC	AMSDOS	max. 251 ext. ROMs
CØØØ					
	Block 2	Block 2			
8ØØØ					
	Block 1	Block 1			
4ØØØ					
	Block Ø	Block Ø	Betriebs-System		
ØØØØ					

1.1.1.1 Speicheraufteilung im CPC

LOW JUMP RST 1

Dieser Befehl dient zum Aufruf einer Routine im Betriebssystem oder im darunterliegenden RAM. Direkt hinter dem RST-Befehl muß die Adresse der aufzurufenden Routine stehen. Da für den Bereich von 0 bis &3FFF 14 Adreßbits ausreichen, benutzt man die oberen beiden Bits für die Auswahl von ROM oder RAM:

Bit 14 = 0 Betriebssystem ausgewählt
Bit 14 = 1 RAM ausgewählt
Bit 15 = 0 BASIC-ROM ausgewählt
Bit 15 = 1 RAM ausgewählt

Ein Aufruf der Betriebssystemroutine &1410 könnte dann so aussehen:

```
RST 1

DW &1410 + &8000
```

Durch das gesetzte Bit 15 ist im Bereich von &C000 bis &FFFF RAM selektiert, während durch das gelöschte Bit 14 das Betriebssystem angesprochen wird.

Der Kode an der Adresse 8 besteht lediglich aus einem Sprung zu &B98A.

SIDE CALL RST 2

Dieser Restart-Befehl dient zum Aufruf einer Routine in einem Expansion-ROM. DerCBefehl wird dann benutzt, wenn ein Programm, daß als ROM-Modul vorliegt, mehr als 16 KByte benötigt und nicht mehr in einem Expansion-ROM Platz hat. Dann kann mit Hilfe des SIDE CALL eine Routine im zweiten, dritten oder vierten zugehörigen ROM aufgerufen werden, ohne daß man die absolute Nummer des jeweiligen ROMs kennen muß. Nach dem RST 2-Befehl muß die Adresse der Routine - &C000,

d.h. also die relative Adresse bezogen auf den Start des ROMs, stehen. Die obersten beiden Bits werden zur Auswahl der vier verschiedenen ROMs benutzt.

An Adresse &0010 steht ein Sprung zu &BA1D.

FAR CALL RST 3

Mit Hilfe dieses RST-Befehls können Sie eine Routine irgendwo im ROM oder RAM aufrufen. Dazu muß hinter dem RST 3-Befehl die 2-Byte-Adresse eines Parameterblocks stehen, der aus drei Bytes besteht. Diese ersten beiden Bytes enthalten die Adresse der Routine, die aufgerufen werden soll, und das dritte Byte muß den gewünschten ROM/RAM-Status enthalten. Dabei wird durch die Werte von 0 bis 251 das entsprechende Zusatzrom angesprochen. Die verbleibenden vier Werte haben folgende Funktion:

Wert	&0000-&3FFF	&C000-&FFFF
252	Betriebssystem	BASIC
253	RAM	BASIC
254	Betriebssystem	RAM
255	RAM	RAM

An Adresse &0018 steht ein Sprung nach &B9C7.

RAM LAM RST 4

Mit Hilfe dieses RST-Befehls können Sie von einem Maschinenprogramm den Inhalt des RAMs lesen, unabhängig vom jeweils gewählten ROM-Zustand. Der RST 4-Befehl ersetzt dabei den Befehl

```
LD A,(HL)
```

HL muß dazu also die Adresse der zu lesenden Speicherzelle enthalten. An Adresse &0020 steht ein Sprung zu &BAD6.

FIRM JUMP RST 5

Mittels dieses RST-Befehls kann man zu einer Routine im Betriebssystem springen. Die Adresse muß dabei unmittelbar auf den RST 5-Befehl folgen. Das Betriebssystem-ROM wird enabled, bevor die Routine angesprungen wird und wird bei der Rückkehr wieder disabled. An Adresse &0028 steht ein Sprung zu &BA35.

1.2 Der Prozessor Z80

In den frühen 70er Jahren begann der Siegeszug der Mikroprozessoren. Die Firma INTEL konnte mit dem Prozessor 8080 einen bedeutenden Marktanteil erreichen, da zum Zeitpunkt der Markteinführung in dieser Klasse praktisch keine Konkurrenz vorhanden war. Dies macht sich allerdings auch bemerkbar, wenn die Leistungsdaten des Prozessors genauer untersucht werden. So benötigt der 8080 noch drei verschiedene Betriebsspannungen und zwei weitere ICs zur Steuersignalerzeugung und Taktgenerierung.

In den Jahren 74/75 wurde von der Firma ZILOG der Z80 entwickelt. Anstatt aber einen von Grund auf neuen Prozessor zu entwickeln, hielt man sich an das so gut angekommene Konzept des 8080. Aus diesem Grunde ist der Z80 zum 8080 aufwärts-kompatibel, d.h. alle für einen 8080 geschriebenen Programme laufen auch auf einem Z80-Prozessor. Allerdings wurden alle mittlerweile beim 8080 als ungünstig erkannten Eigenschaften beseitigt und der Befehlssatz wurde stark erweitert. Auch benötigt der Z80 nur eine Betriebsspannung von +5Volt und aufwendige externe ICs zur Steuersignalerzeugung sind überflüssig.

Doch betrachten wir im Telegrammstil die Leistungsdaten des Prozessors, bevor wir auf seine Eigenschaften etwas konkreter eingehen.

Einfache Stromversorgung 5 Volt
Einfacher Takt
TTL-Kompatibel
Wahlweise 2.5, 4, 6 oder sogar 8 MHz Taktfrequenz
Softwarekompatibel mit 8080
Doppelter Registersatz, zusätzlich zwei Indexregister
Nicht maskierbarer Interrupteingang
Maskierbarer Interrupteingang mit drei Betriebsarten
Selbstätiger Refresh von dynamischen Rams
8080-Peripherie-ICs direkt anschließbar

A 11	1		A 10
A 12			A 9
A 13			A 8
A 14			A 7
A 15			A 6
Ø			A 5
D 4			A 4
D 3			A 3
D 5			A 2
D 6			A 1
+5V			A 0
D 2			GND
D 7			RFSH*
D 0			M1*
D 1			RESET*
INT*			BUSRQ*
NMI*			WAIT*
HALT*			BUSAK*
MREQ*			WR*
IORQ*			RD*

1.2.1.1 Pinout des Z80

Diese Leistungsdaten und die große Menge an fertiger Software haben den Z80 zu einem der erfolgreichsten 8-Bit-Prozessoren werden lassen.
Im Bereich der Home- und Personalcomputer hat nur ein weiterer Prozessor, der 6502, eine vergleichbare Verbreitung gefunden.

1.2.1 Die Anschlüsse des Z80

Nach diesem kurzen Überblick über die Leistungsmerkmale wollen wir zunächst die Belegung der 40 Pins des Z80 betrachten.

Die Anschlüsse des Z80 lassen sich in die vier Gruppen Datenbus, Adressbus, Steuerbus und Versorgungsleitungen zusammenfassen.

Adressbus

A0 - A15 :Adresslines

Über diese Anschlüsse wird eine Speicherzelle im Adressbereich angewählt. Der Adressbereich umfasst 65536 Speicherplätze. Bei der Behandlung der I/O-Befehle werden die unteren 8 Adressbits benutzt, um die entsprechende I/O-Adresse auszugeben. Somit sind 256 verschiedene Ports möglich. Mit gewissen Einschränkungen im Befehlssatz können aber sogar 65536 Ports adressiert werden. Dann werden alle 16 Adressleitungen zur Bildung der Portadresse herangezogen. Auf diesen Spezialfall werden wir später zurückkommen.

Datenbus

D0 - D7 :Datalines

Über diese bidirektionalen Leitungen gelangen die Daten von und zum Prozessor. Sie stellen die Verbindung zwischen Prozessor und der durch den

Adressbus ausgewählten Speicherzelle oder auch Portadresse her.

Steuerbus

M1* :Machine Cycle One

Dieses Steuersignal zeigt an, daß der Prozessor den Operationscode vom Datenbus liest. Der Stern deutet übrigens bei diesem und den folgenden Signalen an, daß es sich hierbei um lowaktive Signale handelt.

MREQ* :Memory REQuest*

Dieses Ausgangssignal zeigt durch ein Low an, daß der Prozessor einen Schreib- oder Lesezugriff auf eine Speicheradresse vornimmt und die Adresse auf dem Adressbus gültig ist.

IORQ* :Input/Output ReQuest*

Ein Low dieses Ausgangs zeigt an, daß der Prozessor einen Schreib- oder Lesezugriff auf eine Portadresse vornimmt und die Portadresse auf dem Adressbus gültig ist.

RD* :ReaD*

Dieses Ausgangssignal ist Low, wenn der Prozessor Daten aus einer Speicherzelle oder Portadresse lesen will. Durch Verknüpfung mit MREQ* und IORQ* kann zwischen Lesen aus Speicher und Port unterschieden werden.

WR* :WRite*

Dieses Signal des Z80 wird Low, wenn bei Schreibzugriffen des Z80 auf Speicher oder Portadressen die Daten auf dem Datenbus gültig sind. Auch hier kann wieder durch Verknüpfen des WR* mit MREQ* und IORQ* unterschieden werden, ob Daten in den Speicher oder eine Portadresse geschrieben werden.

RESET* :

Wird dieser Eingang auf Low gelegt, dann wird der Programmzähler mit dem Wert &0000 geladen, Interrupts werden gesperrt und der Interruptmodus 0 wird eingeschaltet. Sobald der Eingang wieder High wird, beginnt der Prozessor das Programm ab der Adresse &0000.

NMI* :Non Maskable Interrupt*

Durch eine High-Low-Flanke an diesem Eingang wird der Prozessor immer im laufenden Programm unterbrochen. Der Programmzähler wird mit den in den Adresse &0066 und &0067 gespeicherten Werten geladen und an dieser Stelle wird das Programm fortgesetzt.

IRQ* :Interrupt ReQuest*

Durch ein Low an diesem Eingang kann der Prozessor im laufenden Programm unterbrochen werden, wenn diese Art des Interrupt per Befehl freigegeben ist. Die Auswirkungen unterscheiden sich je nach gewähltem Interruptmodus und werden später besprochen. IRQ* stellt im Gegensatz zu NMI* ein statisches Signal dar und muß bis zum Erkennen der Interruptanforderung anliegen.

WAIT* :

Mit Hilfe dieses Signals kann der Lese- oder Schreibzugriff des Z80 an langsamere Speicher oder spezielle Bedingungen des Systems angepasst werden.

BUSRQ* :BUSReQuest*

Wird dieser Eingang Low, dann werden nach der Abarbeitung des laufenden Befehls Adress- und Datenleitungen sowie alle Ausgangssteuerleitungen hochohmig und das BUSAK*-Signal wird Low. Jetzt könnte ein zweiter Prozessor den Zugriff auf den Speicher und die Peripheriebausteine übernehmen, hauptsächlich wird dieses Signal jedoch für DMA

benutzt (DMA=Direkt Memory Access, sehr schneller Datentransfer bei Umgehung des Prozessors).

BUSAK* :BUSAKnowledge*

BUSAK* stellt das mit BUSRQ* korrespondierende Ausgangssignal dar. Ein Low zeigt dem DMA-Controller oder zweiten Prozessor an, daß alle Steuer- und Bussignale hochohmig sind und ein Zugriff jetzt erfolgen kann.

HALT* :

Dieser Ausgang wird Low, nachdem der Prozessor den Maschinensprache-Befehl HALT ausgeführt hat. Nach diesem Befehl 'tut' der Prozessor nichts mehr, er führt NOPs aus, um den Refresh sicherzustellen. Nur ein Interrupt kann ihn wieder 'wecken'.

RFSH* :ReFreSH*

Dieses Ausgangssignal zeigt an, daß auf den unteren sieben Adressleitungen eine gültige Refresh-Adresse liegt. Da der Prozessor nur zu bestimmten Zeiten den Adress- und Datenbus benötigt, kann in der verbleibenden Zeit der Adressbus zum Auffrischen dynamischer Rams verwendet werden, ohne daß aufwendige Elektronik oder spezielle Auffrisch-Routinen benötigt werden.

Takt und Stromversorgung

0 :Phi

Der Eingang Phi liefert den Takt für den Prozessor. Da der Z80 ein statisches IC ist, kann der Takt von 0 Hertz bis zur angegebenen Maximalfrequenz betragen. Allerdings werden an die Form des Taktsignals bestimmte Anforderungen gestellt. Laut Datenblatt darf die maximale Lowzeit dieses Signals 2 Mikrosekunden betragen. Dieser Wert ist allerdings mehr von akademischem Interesse, da man ja bemüht sein wird, den Prozessor mit möglichst

hoher Taktfrequenz zu versorgen, um ein schnelles Abarbeiten des Programms zu erhalten.

GND :

Masseanschluß des Prozessors.

Vcc :

Über diesen Anschluß bekommt der Z80 seinen Saft, sprich +5 Volt Gleichspannung und ca. 150 bis 200 Milliampere.

1.2.2 Der Register-Aufbau des Z80

Wie schon zu Beginn erwähnt, ist der Z80 so konstruiert worden, daß Programme des 8080 ohne weiteres übernommen werden können. Allerdings ist die Anzahl der Register des Z80 deutlich höher.

Aber was ist eigentlich ein Register?

Nun, ein Register ist nichts anderes als ein Schreib/Lese-Speicher auf dem Prozessorchip. Jeder Prozessor muß eine Mindestzahl von Registern aufweisen. In diesen Speicherzellen werden Daten gespeichert und die Ergebnisse von arithmetischen und logischen Befehlen abgelegt. Andere Register haben spezielle Aufgaben, wie die Verwaltung des Stack oder werden als Programmzähler verwendet.

Da Operationen wie ein Transfer von Daten zwischen zwei Registern oder die Addition zweier Registerinhalte nicht über den Datenbus abgewickelt werden, kann eine solche Operation sehr viel schneller durchgeführt werden, als wenn die benötigten Werte aus externen Speicherplätzen geholt werden müssen.

Als grobe Regel kann man sagen, daß Prozessoren mit vielen Registern denen mit weniger internem Speicher bei der Bearbeitung gleicher Programme überlegen sind, da der Datentransfer innerhalb des Prozessors immer schneller ist, als ein Transfer von und zu externen Speicherplätzen.

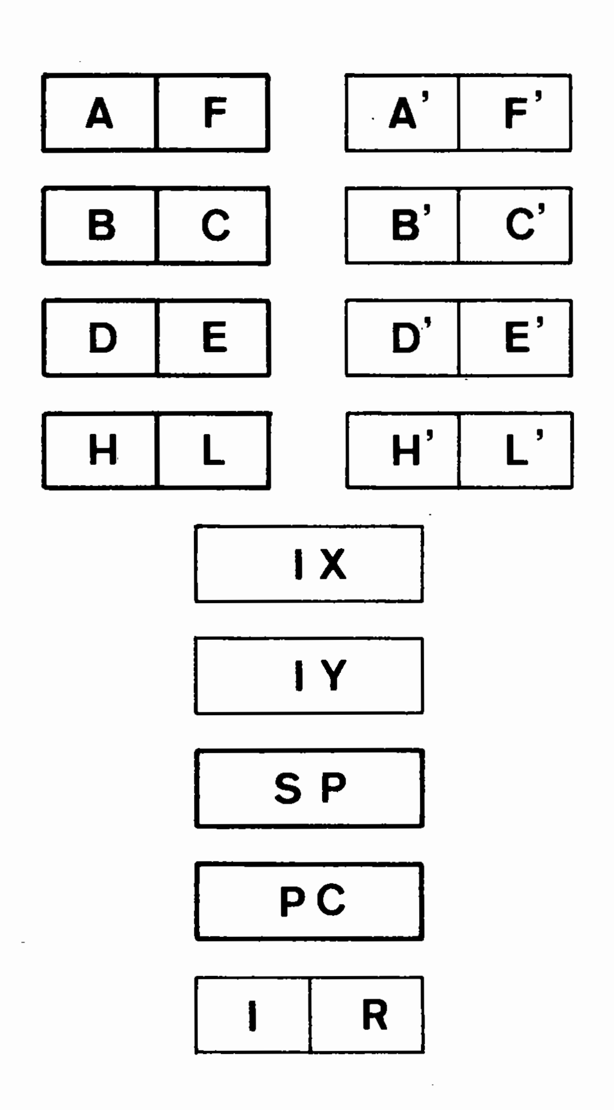
Insgesamt verfügt der Z80 über 22 Register, 18 Register mit 8 Bit und vier 16-Bit-Register. Die Aufteilung zeigt die Grafik 1.2.2.1.

In der Grafik fallen einige Register durch ihre stärkere Umrahmung auf. Diese Register sind auch im 8080 enthalten. Auch ist auffällig, daß die meisten 8-Bit-Register doppelt vorhanden sind. Dies sind die Register A, F, B, C, D, E, H und L. Der Z80 stellt sie in doppelter Ausführung zur Verfügung und der Programmierer kann per Befehl zwischen den beiden Sets wählen.

Wir werden zukünftig nur von einem Registersatz sprechen. Das ist beim CPC auch insofern richtig, da ohne spezielle Tricks dem Programmierer beim CPC sowieso nur ein Registersatz zur Verfügung steht. Der alternative Registersatz wird vom Betriebssystem für die Interruptsteuerung benutzt. Merken Sie sich aber, daß alle Aufgaben eines Registersatzes auch vom alternativen Registersatz übernommen werden können, wenn dieser nicht für spezielle Zwecke belegt ist.

Die Register B bis L stellen allgemein verfügbare 8-Bit-Register dar, während den Registern A und F besondere Aufgaben zukommen.

Das A-Register wird allgemein als Akku oder Akkumulator bezeichnet. Im Akku erhält man das Ergebnis von allen arithmetischen und logischen Operationen mit 8-Bit-Format. Auch muß im Akku bei diesen Operationen ein Operand gespeichert sein. Um z.B. zwei Bytes zu addieren, ist es nötig, einen Operanden in den Akku zu speichern, der andere Operand kann in einem anderen Register oder außerhalb des Prozessors im Speicher untergebracht sein. Nach der Addition steht dann das Ergebnis im Akku.
Da bei diesen Aufgaben das Ergebnis so groß werden kann, daß es mit nur 8 Bit nicht mehr ausgedrückt werden kann (255 + 255 = 510), wird ein weiteres Bit benötigt, um das Ergebnis



1.2.2.1 Registersatz Z 80

korrekt darzustellen. Diese Aufgabe wird vom F-Register übernommen. Das F-Register, allgemein als Flag-Register bezeichnet, ist in einzelne Bits aufgeteilt. Eines dieser Bits hat (unter anderem) die Aufgabe, einen evtl. Übertrag (engl. Carry) solcher Additionen zu bewahren. Andere Bits zeigen, ob das Ergebnis von Rechenoperationen oder Vergleichen gleich Null ist u.s.w.

Die Register B bis L können aber nicht nur einzeln angesprochen werden. Jeweils B und C, D und E sowie H und L können zu 16-Bit-Registern zusammengefasst werden. Diese Doppelregister haben dann sinnvollerweise den Namen der beiden Einzelregister, als BC, DE und HL. Doppelregister eignen sich hervorragend zum Adressieren von Tabellen und zum Transportieren und Durchsuchen von Datenblöcken.
Dem HL-Doppelregister kommt noch eine besondere Bedeutung zu. Da der Z80 mit Befehlen zur Addition und Subtraktion von 16-Bit-Werten ausgestattet ist, fungiert das HL bei solchen Anweisungen als 16-Bit-Akku.

Nur mit 16-Bit-Werten arbeiten PC, SP, IX und IY (Anm.: Spezialisten wissen, daß die Möglichkeit besteht, die Index-Register auch byte-weise zu manipulieren, wir werden IX und IY aber als reine 16-Bit-Register betrachten).

PC ist der Programm Counter oder Programm-Zähler. Der Inhalt des PC wird als Adresse für externe Speicher auf den Adressbus gelegt. Mit jedem Befehl wird der PC automatisch inkrementiert (um eins erhöht). Bei Befehlen mit mehr als einem Byte wird der PC automatisch um die benötigte Anzahl erhöht. Werden in einem Programm Sprünge notwendig, dann wird die neue Programmadresse automatisch in den PC geladen, und der Prozessor arbeitet ab dieser Adresse weiter.

SP ist der sogenannte Stackpointer. Der Stack (oder Stapel) wird benötigt, wenn von einem Programm Unterprogramme aufgerufen werden. In diesem Fall wird automatisch die Rücksprungadresse auf dem Stack abgelegt und nach Beendigung des Unterprogramms in den PC zurückgeladen.

Die beiden 16-Bit-Register IX und IY ermöglichen durch spezielle Befehle ein besonders wirkungsvolles Arbeiten mit Tabellen.

Bleiben noch die beiden Register I und R. Das I-Register oder Interrupt-Register wird in Verbindung mit der speziellen Interrupt-Betriebsart IM3 verwendet. In diesem Modus muß der den Interrupt erzeugende Baustein auf Anforderung des Prozessors einen 8-Bit-Wert liefern. Dieser Wert als Low-Byte und der Inhalt des I-Registers als High-Byte bilden die Adresse der Interrupt-Routine.
Das R- oder Refresh-Register wird in Verbindung mit dem vom Z80 automatisch durchgeführten Refresh benötigt. Nach jedem Holen eines Befehls werden die untersten sieben Bits dieses Registers automatisch incrementiert. Das achte Bit verbleibt immer, je nach Programmierung, Null oder Eins.
Sowohl I- wie R-Register werden im CPC nicht verwendet. Da über den Zustand des R-Registers keine Aussage gemacht werden kann und sich der Wert ständig ändert, kann dieses Register als Zufallsgenerator benutzt werden.

1.2.3 Besonderheiten des Z80 im CPC

Die vielfältigen Möglichkeiten des Z80 lassen den Hard- und Software-Designern freie Hand bei der Konstruktion eines Computers. Diese CPU (Central Processing Unit) kann gleichermaßen effektiv in Minimalsystemen und solch leistungsstarken Geräten wie den CPC-Rechnern eingesetzt werden.
Gerade die Entwickler des CPC haben tief in die Trickkiste gegriffen, um mit einem Minimum an Bauteilen ein Maximum an Leistung zu erreichen. Dabei sind zwangsläufig einige Besonderheiten entstanden, deren Kenntnis für eine effektive Programmierung und Nutzung des Gerätes besonders in Maschinensprache wichtig sind. Diese Spezialitäten sollen jetzt unter die Lupe genommen werden.

Da wäre zunächst die Interruptsteuerung des CPC.
Einzige Interrupt-Quelle im CPC ist das Gate Array, dieser

Die Notwendigkeit dieses Anschlusses am Z80 resultiert noch aus der Zeit, als die verfügbaren Speicher-ICs recht gemütliche Gesellen waren. Besonders die ersten EPROMs ließen sich nach Anlegen der Adresse bis zu einer Microsekunde Zeit, bis sie die Daten parat hatten.
Um den Z80 mit solchen 'Langweilern' zu betreiben, war es nötig, eine bestimmte Zeit zu warten. Diese Wartezeit kann durch das Signal WAIT* erzeugt werden. Nach jeder negativen Flanke am Takt-Eingang überprüft der Prozessor den Zustand des Wait*-Anschlusses. Liegt dieser Anschluß auf 0 Volt, dann fügt der Z80 einen sogenannten Wait-Zyklus von der Dauer des Taktes ein. Nach Ablauf des Taktsignals, also mit der negativen Flanke, wird wieder der Zustand der Wait*-Leitung geprüft u.s.w.
Das im CPC dies Signal verwendet wird, liegt aber nicht an den verwendeten Speicher-ICs. Die sind allemal schnell genug für einen Z80 mit 4 MHz. Der Grund ist die nötige Synchronisation zwischen Prozessor und Video-Controller. Da beide ICs auf den Speicher zugreifen können, muß eine Kontrolle darüber bestehen, wer zu welcher Zeit an der Reihe ist. Dabei hat der Video-Controller unbedingten Vorrang, da sonst die Anzeige auf dem Monitor stark gestört werden könnte. Um diese Synchronisation zu erreichen, wird zu jedem vierten Taktsignal für den Prozessor ein Wait*-Signal erzeugt. Obwohl der Prozessor mit 4 MHz (Mega Hertz = Millionen Schwingungen pro Sekunde) angesteuert wird, ergibt sich durch die Waitzyklen eine effektive Arbeitsfrequenz von ca. 3.3 MHz.

Die Signale BUSRQ* und BUSAK*, die Steuersignale für den DMA-Betrieb, sind im CPC nicht benutzt. Wohl aber sind sie auf den Expansion Connector geführt und stehen hier externen Erweiterungen zur Verfügung.
Auch das Signal HALT* wird im CPC nicht verwendet, ist aber ebenfalls am Expansion Connector verfügbar.

1.3 Das Gate Array, der System-Koordinator

Fast alle Bauteile im CPC sind handelsüblich. Man kann sie in jedem gut sortierten Elektronik-Laden erwerben. Die einzigen Ausnahmen sind die ROMs und das Gate Array. Das zuletzt genannte IC soll uns in diesem Abschnitt beschäftigen.
Dieses 40-polige IC ist speziell für den CPC entwickelt worden und hat mehrere wichtige Aufgaben. Wollte man alle integrierten Funktionen mit TTL-Gattern nachbilden, die Anzahl der ICs des CPC würde sich schnell mehr als verdoppeln.
Die Aufgaben des Arrays sind unter anderem:

Erzeugung aller benötigten Taktfrequenzen
Erzeugung der Signale für den Betrieb des dynamischen Rams
Steuerung der Zugriffe auf das Ram
Ab- und Zuschalten des ROM in den Speicherbereich
Erzeugung der Video-Signale
Erzeugung der RGB-Informationen für den Farb-Monitor
Steuerung des Bildschirmmodus
Speicherung der Tinten-Farben
Erzeugung des Interrupt-Impulses

Leider sind über dieses interessante IC nur sehr wenig Informationen verfügbar. Ein Datenblatt oder eine vergleichbare Beschreibung sind nirgends erhältlich, da der Hersteller das Innenleben wohl als Betriebsgeheimnis ansieht.
Unsere Bemühungen und Versuche, die Funktion des ICs möglichst gründlich zu erforschen, waren allerdings recht erfolgreich. So wollen wir Ihnen unsere Ergebnisse nicht vorenthalten.

1.3.1 Die Anschluß-Belegung des Gate Array

Bevor wie die auf die Funktionen der einzelnen Anschlüsse des Gate Array eingehen können, muß eine Information vorweggeschickt werden. Mittlerweile existieren mindestens

Pin	Left	Right
1	CPU ADDR*	MA0/CCLK
	READY	Ø
	CAS*	Vcc1
	244EN*	RESET*
	MWE*	R
	CAS ADDR*	GND
	RAS*	G
	XTAL	Vcc2
	Vcc2	B
	INTERRUPT*	D 7
	SYNC*	D 6
	ROMEN*	D 5
	RAMRD*	D 4
	HSYNC	D 3
	VSYNC	D 2
	IORQ*	D 1
	M1*	D 0
	MREQ*	DISPEN
	RD*	Vcc1
	A 15	A 14

1.3.1.1 Pinout des Gate Array 40007 & 40008

D 5 (1)	D 4
D 6	D 3
D 7	D 2
CCLK	D 1
SYNC*	V_{SS}
V_{DD}	D Ø
RESET*	RAS*
B-Ausgang	MWE*
DISPEN	INT*
C-Ausgang	CAS ADDR*
HSYNC	A 14
R-Ausgang	RAM RD*
YSYNC	A 15
CPU*	ROMEN*
V_{SS}	V_{SS}
CAS*	V_{DD}
MREQ	CK 16
IORQ*	244 EN*
PHI	READY
NMI	RD*

1.3.1.2 Pinout des Gate Array 40010

drei Versionen des Gate Attay. Im CPC 464, dem ersten CPC-Rechner trug das IC die Bezeichnung 40007. Dieses IC wurde im Betrieb dermaßen heiß, daß es notwendig wurde, das IC durch ein aufgeklemmtes Aluminiumblech zu kühlen. Dieser Zustand war auf Dauer nicht praktikabel, da auch mit zusätzlicher Kühlung das IC durch Überhitzung zerstört werden konnte. Im CPC 664 wurde das IC 40008 eingesetzt. Durch Änderungen im internen Aufbau konnte die umgesetzte Verlustleistung reduziert. Diese Version wurde nur noch geringfügig warm. Im CPC 6128 schließlich ist das IC 40010 eingesetzt. Bei diesem IC wurde die Reihenfolge der Anschlüsse geändert. Wahlweise können aber auch die älteren Versionen des Gate Array im 6168 eingesetzt werden. Der benötigte Platz auf der Leiterplatte ist vorhanden.
Bei der Beschreibung der Anschlüsse haben wir uns an das Pinout der im CPC 6128 eingesetzten Version gehalten. Die in Klammern angegebenen Anschluß-Nummern beziehen sich auf die älteren im 664 und 464 eingesetzten Versionen des GA.
Das alles bestimmende Signal des CPC ist das am Pin 24 (Pin 8) (XTAL) anliegende Quarzsignal mit einer Frequenz von 16 MHz. Diese Frequenz wird von einer mit zwei Gattern aufgebauten typischen Oszillatorschaltung erzeugt und stellt quasi den Her(t)zschlag des CPC dar.

Die durch vier geteilte Eingangsfrequenz steht als Taktsignal von 4 MHz am Pin 19 (Pin 39) als Takt Phi für den Prozessor zur Verfügung.

Ein weiteres Mal durch vier geteilt ergibt sich eine Frequenz von 1 MHz. Dieses Signal wird am Pin 14 (Pin 1) des Gate Array zur Verfügung gestellt.
Das 1-MHz-Signal hat zwei Verwendungen. Zum einen ist es das Taktsignal für den Sound Chip, zum anderen bestimmt es mit, ob der Prozessor oder der CRTC das RAM adressieren kann. Bei einem Low werden die Adressleitungen des Prozessors über die Multiplexer-ICs 74LS153 zum RAM geschaltet.
Da allerdings die Ansteuerung des RAM im CPC nicht ohne Tücken ist, finden Sie eine ausführliche Beschreibung der Ram-Steuersignale in einem späteren eigenen Kapitel.

Da die Ram-Bausteine nur über 8 Adressleitungen verfügen, muß die gesamte 16-Bit-Adresse gemultiplext, also zeitlich nacheinander an die Eingänge gelegt werden. Diese zeitliche Steuerung wird mit den Signalen CAS ADDR* Pin 31 (Pin 6), CAS* Pin 16 (Pin 3) und RAS* Pin 34 (Pin 7) erreicht. Die Signale RAS* und CAS* werden direkt an die RAMs gelegt, das Signal CAS ADDR* wird an die schon erwähnten Multiplexer geführt.

Auch das Signal MA0/CCLK an Pin 4 (Pin 40) des Gate Array hat eine Frequenz von 1 MHz. Dieses Signal ist allerdings zum CPU ADDR*-Signal phasenverschoben, d.h. die beiden Frequenzen sind zu unterschiedlichen Zeiten High. MA0/CCLK hat ebenfalls eine Doppelfunktion. Einmal stellt es das Taktsignal für den CRTC dar, der von diesem Signal alle weiteren Signale ableitet, zum zweiten wird es als Hilfsadressbit an einen der vier Adress-Multiplexer gelegt. Die Funktion dieses Hilfsadressbits wird ebenfalls später bei der Ansteuerung der RAM genauer erörtert werden.

Des weiteren wird vom Gate Array das Signal RAMRD* am Pin 29 (13) erzeugt. Dieser Anschluß wird dann Low, wenn der Prozessor nach Anlegen einer Adresse Daten aus dem RAM lesen will und dies durch sein RD*-Signal dem Array an Pin 21 (19) mitteilt. Da sich in weiten Bereichen ROM und RAM überlagern, kann das RD*-Signal des Prozessors nicht direkt verwendet werden. Sollen Daten aus dem ROM gelesen werden, so bleibt das Signal RAMRD* High und die Ausgänge des DATA LATCH/BUFFERs 74LS373 werden hochohmig. Dadurch kann zu diesen Zeiten keine Information vom RAM auf den Datenbus gelangen, obwohl die Speicheradresse auch an das RAM gelangt ist und es an seinen Ausgängen ein Byte bereitstellt.
Zusätzlich zum RAMRD* wird das READY-Signal vom Pin 22 (Pin 2) des GA an das IC 74LS373 gelegt. Dieses Signal erzeugt am Prozessor das Signal zum Einfügen der Wait-Zyklen. Durch die zusätzliche Verschaltung des READY mit dem Latch/Buffer wird erreicht, daß sich die Information auf dem Prozessor-Datenbus während der Wait-Zyklen nicht ändert. Der 74LS373 speichert nach Anlegen eines High am Pin 11 die

derzeitige Ausgangsinformation, bis dieser Anschluß wieder Low wird. Danach verhält sich das IC wie ein einfacher Puffer, d.h. die Ausgänge folgen den Änderungen der Eingänge unmittelbar.
Das Signal ROMEN* an Pin 27 (12) des GA wird Low, wenn der Prozessor Daten aus dem ROM lesen will. Das im CPC eingebaute 32k-BASIC- und Betriebssystem-ROM belegt die Adress-Bereiche von &0000 bis &3FFF und von &C000 bis &FFFF. Es ist also in zwei unabhängigen Hälften ansprechbar. Ob in den sich überlagernden Speicherbereichen aus ROM oder RAM gelesen werden soll, muß dem GA über einen OUT-Befehl mitgeteilt werden. Dabei ist es durchaus möglich, nur eine Hälfte des ROM zu aktivieren.
Entsprechend der gewünschten Speicherkonfiguration dekodiert der GA den Zustand der Adressleitungen A14 und A15. Je nach gefordertem Speicher wird dann beim Lesen das RAMRD*- oder ROMEN*-Signal aktiv.

Ein Schreibbefehl des Prozessors geht unabhängig von der gewählten Speicherkonfiguration immer ins Ram. Dazu wird vom GA das Signal MWE* erzeugt.

Zusätzlich zur beschriebenen Funktion werden die Adressleitungen A14 und A15 an den Pins 28 (20) und 30 (21) aber noch für einen anderen Zweck verwendet. Auch das GA hat eine Portadresse, die benutzt wird, um die verschiedenen Möglichkeiten des GA zu programmieren. Die Portadresse ist &7F00 und wird über die Adressleitungen (A14 High, A15 Low) und das Signal IORQ* an Pin 17 (Pin 18) decodiert.

Da der Datenbus des Z80 nicht direkt mit den Datenleitungen D0 bis D7 des GA verbunden ist, legt das Array den Anschluß 244EN* auf Low, wenn die Portadresse &7F00 in der zuvor beschriebenen Art erkannt wird. Dadurch werden die Ausgänge 74LS244 (ein Datenbuspuffer) freigegeben und das vom Z80 gelieferte Byte kann in das Array geschrieben werden.

Aber auch das Signal IORQ* hat für den GA eine Doppelbedeutung. Eine spezielle Eigenart des Z80 ist es, bei

einem erkannten Interrupt gleichzeitig die Signale IORQ* und M1* auf Low zu legen. Dieser Zustand wird vom GA erkannt und der Interrupt-Impuls wird sofort gelöscht. Ist dagegen durch den Befehl DI, Disable Interrupt, die Behandlung des IRQ ausgeschaltet, so bleibt der Anschluß 32 (10) des GA bis zum Wiederzulassen des IRQ Low. Sobald der IRQ mit EI wieder eingeschaltet ist, wird der anliegende Interrupt erkannt und der Interrupt-Ausgang wird wieder High.
Erzeugt wird das Interrupt-Signal an Pin 32 (Pin 10) durch eine programmierbare Teilerkette im GA. Diese Teilerkette wird mit dem CRTC-Signal HSYNC versorgt und teilt die anliegende Frequenz durch 52. Da der HSYNC-Impuls ca. alle 65 Mikrosekunden auftritt, ergibt sich eine Zeit von 3,3 Millisekunden zwischen zwei Interruptimpulsen. Dabei werden die Impulse mit dem VSYNC-Signal des CRTC gekoppelt. Die Breite des VSYNC ist im CRTC auf ca. 500 Mikrosekunden programmiert. Nach etwa 125 Mikrosekunden erscheint der Interrupt, somit bleibt der Interrupt-Routine noch etwa 375 Mikrosekunden Zeit, am Portbit 0 des Port B des 8255 zu prüfen, ob ein VSYNC vorhanden ist. Dieses Signal wird als Zeitgeber bei verschiedenen Operationen benutzt.
Dieser Fall tritt aber nur bei jedem fünfzehnten Interrupt auf, bei den restlichen 14 Abfragen ergibt sich ein High des VSYNC, der interne Zähler wird nicht beeinflußt.

Die Signale HSYNC und VSYNC werden aber natürlich wie auch DISPEN zum Erzeugen des Video-Signals benötigt. Eine Verknüpfung dieser Signale ergibt das SYNC*-Signal am Pin 5 (Pin 11) des GA.

1.3.2 Der Registeraufbau des Gate Array

Um alle beschriebenen Aufgaben ausführen zu können, müssen Daten im GA gespeichert werden. Die genaue Anzahl der internen Register ist nicht bekannt, allerdings können wir die vermutlich wichtigsten Register beschreiben.

Wie auch alle anderen Bausteine im CPC wird das GA über die Port-Adressierung angesprochen.

Die belegte Adresse ist &7Fxx. Daraus resultiert, daß das Adressbit A15 Low, das Adressbit A14 dagegen High sein muß. Die übrigen Adressbits (A12 bis A8) müssen gesetzt (auf High-Pegel) sein, da die anderen Peripherie-Bausteine in ähnlich unvollständiger Weise decodiert werden. Bei diesen Bausteinen sind die Selektionseingänge auch nur mit einzelnen Adressbits verbunden.
Der Zustand des unteren Adressbytes ist für die Dekodierung unerheblich, hier kann jeder beliebige Wert anliegen.

Insgesamt kann zwischen drei verschiedenen Registern unterschieden werden.
Die ersten beiden Register stehen im Zusammenhang mit der Farberzeugung, genauer mit den durch PEN und INK festgelegten Farbzuordnungen.
Das erste Register wird mit der Adresse geladen, in die ein Farbwert eingeschrieben werden soll. Wir wollen es weiterhin als Farbnummer-Register (FN-Reg) bezeichnen.
Den eigentlichen Farbwert kann man danach in das zweite Register (unter derselben Portadresse!) einschreiben. Dieses Register werden wir als Farbwert-Register (FW-Reg) bezeichnen.

Das dritte Register ist ein Multifunktionsregister (MF-Reg) und bestimmt den Bildschirmmodus und die Speicherkonfiguration. Dabei wird die Auswahl der verschiedenen Möglichkeiten durch die einzelnen Bits innerhalb des Registers bestimmt.

Alle Register des GA lassen sich nur beschreiben. Ein Auslesen der Werte ist NICHT möglich.

Da das GA nur über eine einzige Portadresse angesprochen werden kann, muß es einen Weg geben, zwischen den verschiedenen Gruppen zu unterscheiden. Diese Unterscheidung wird mit den beiden obersten Bits des Datenbytes festgelegt.

Die möglichen Kombinationen lauten:

Bit7	Bit6	Funktion
0	0	Wert in das FN-Reg schreiben
0	1	Farbwert in das gewählte FW-Reg schreiben
1	0	Wert in das MF-Reg schreiben
0	0	Für Speicherumschaltung im 6128 genutzt

Was hat es aber nun mit den Farbnummer- und Farbwert-Registern auf sich?
Im Grunde stellen diese beiden Register die Entsprechungen zu den Basic-Befehlen PEN und INK dar. Der PEN-Befehl wird bekanntermaßen benutzt, um die aktuelle Schreibfarbe auf dem Monitor zu verändern. Die Zuordnung einer PEN-Nummer zu einer Farbe kann mit dem INK-Kommando festgelegt werden. Dazu wird die zu ändernde Nummer und der gewünschte Farbwert angegeben. Genau diese Funktionen werden mit den beiden Registern ausgeführt. In das FN-Register wird die Nummer der zu ändernden Farbe eingetragen, danach kann der gewünschte Farbwert in das GA geschrieben werden.
Um z.B. die zu PEN 1 gehörende Farbe zu ändern, ist der folgende Ablauf nötig:

OUT &7F00,&X00000001 : OUT &7F00,&X010XXXXX

Im ersten OUT-Befehl sind die Bits 6 und 7 = 0. In den Bits 0 bis 3 wird die Nummer der zu ändernden Farbe angegeben. In unserem Beispiel ist das die Nummer 1. Das Bit 5 hat keine Funktion, das Bit 4 hat eine besondere Bedeutung, auf die wir gleich noch zu sprechen kommen.
Im zweiten OUT-Befehl sind die Bits 6 und 7 so gewählt, daß das FW-Register angewählt ist. Die als 'X' bezeichneten Bits bestimmen nun den Farbenwert. Mit 5 Bit sind zwar 32 verschiedene Farben möglich, aber nur 27 verschiedene Farben werden erzeugt. Die verbleibenden 5 Farben sind mit anderen Farben identisch.

Wenn Sie dieses Beispiel in Basic ausprobieren, werden Sie feststellen, daß sich der gewünschte Erfolg nicht so recht

einstellt. Ein kurzes Aufblitzen der neuen Farbe ist alles, was dabei herauskommt.
Ursache ist eine Eigenart der Software des CPC. Grundsätzlich werden alle Farben 'blinkend' dargestellt. Das bleibt aber unbemerkt, da nicht zwischen verschiedenen, sondern gleichen Farben umgeschaltet wird. Bei jedem Umschalten der Farben werden alle Parameter für das GA neu geladen. Wenn Sie aber vor die OUT-Kommandos den Befehl **'SPEED INK 255,255'** setzen, dann können Sie zumindest bei einigen Versuchen eine deutlich längere Zeit die Auswirkung betrachten.

Doch jetzt die Erklärung des bisher ausgesparten Bit 4 im FN-Reg. Ist dieses Bit beim Zugriff auf das Register gesetzt, dann wird die Information in den Bits 0 bis 3 ignoriert, der im nächsten OUT-Befehl übermittelte Farbwert wird als neue Rahmenfarbe interpretiert.

Das MF-Register wird adressiert, wenn im OUT-Befehl das Bit 7 gesetzt und das Bit 6 Low ist. Die übrigen Bits dieses Registers haben folgende Bedeutung:

```
Bit 5 : Dies Bit hat keine Funktion ?
Bit 4 : 1 = V-Sync-Zähler löschen
Bit 3 : 1 = ROM &C000 bis &FFFF abschalten
Bit 2 : 1 = ROM &0000 bis &3FFF abschalten
Bit 1 : Bildschirm-Modus
Bit 0 : Bildschirm-Modus
```

Über die Funktion des Bit 5 in diesem Register konnte bisher nichts in Erfahrung gebracht werden.

Ist das Bit 4 gesetzt, so wird die Teilerkette für den Interruptimpuls gelöscht und der Zählvorgang der V-Sync-Impulse beginnt von neuem. Auf diese Weise könnte der zeitliche Abstand zwischen zwei Interruptimpulsen verlängert werden. In Basic können Sie sich von der Funktion mittels der folgenden kleinen Programmschleife überzeugen:

```
10 OUT &7F00 , &X10010110 : GOTO 10
```

Nach dem Start ist der Rechner vollständig blockiert. Auch ein Reset über SHIFT/CTRL/ESC ist nicht mehr möglich. In diesem Einzeiler wird das Zähl-Register so schnell gelöscht, daß überhaupt keine Interrupt-Impulse mehr auftreten können. Da aber die Tastatur nur im Interrupt abgefragt wird, hilft nur noch das Aus- und wieder Einschalten, um den CPC wieder bedienbar zu machen.

Die Bits 2 und 3 bestimmen die momentane ROM-Speicherkonfiguration. Ist eins der Bits gesetzt, so befindet sich für den Prozessor in den angegebenen Adressbereichen bei Lesezugriffen das RAM, sind die Bits gelöscht, dann liest der Prozessor die Daten aus dem ROM.
Diese beiden Bits planlos zu manipulieren, führt mindestens zu Fehlermeldungen, Systemabstürze oder ein Reset sind aber genau so möglich.

Die verbleibenden Bits 0 und 1 bestimmen den aktuellen Bildschirm-Modus. Die möglichen Kombinationen sind:

Bit1	Bit0	
0	0	Mode 0, 20 Zeichen/Zeile, 16 Farben
0	1	Mode 1, 40 Zeichen/Zeile, 4 Farben
1	0	Mode 2, 80 Zeichen/Zeile, 2 Farben
1	1	wie Mode 0, aber kein Blinken

Wenn Sie den Einzeiler zum Ausschalten des Interrupt im Mode 1 probiert haben, so werden Sie eine seltsame Veränderung der Zeichen auf dem Bildschirm festgestellt haben. In diesem Beispiel haben wir als Bildschirm-Modus den 80-Zeichen-Modus gewählt und ohne den Bildschirm zu löschen umgeschaltet. Die dargestellten Zeichen sehen aus, als ob in der Mitte jedes Zeichens Punkte fehlen. Die Erklärung zu diesem Phänomen finden Sie im Anschluß an das folgende Kapitel, wenn der Aufbau des Bildschirms und die Darstellung der Zeichen beschrieben wird.

1.4 Der Video-Controller HD 6845

Die Hauptarbeit bei der Erzeugung des Bildes auf dem Monitor leistet der Video-Controller HD 6845, der auch als Cathode Ray Tube Controller, kurz CRTC bezeichnet wird. Dieses IC wurde speziell als Interface zwischen Microprozessoren und Rasterbildschirmen wie den üblichen Monitoren entworfen.
Er erzeugt aus einem einzigen Taktsignal alle für den Monitor erforderlichen Synchronsignale, wobei sich alle benötigten Parameter in weiten Grenzen programmieren lassen.

Bevor wir die Anschlußbelegung und den internen Registeraufbau beschreiben, wollen wir einen kurzen Überblick über die Möglichkeiten dieses interessanten Bausteins geben.

Programmierbare Anzahl der Zeichen pro Zeile
Programmierbare Anzahl der Zeilen pro Bildschirm
Programmierbare vertikale Punktmatrix der Zeichen
Zugriff auf einen Speicherbereich von 16 K
Automatischer Refresh bei Verwendung dynamischer Rams
Cursor-Control-Funktionen
Programmierbarer Cursor (Höhe und Blinken)
Light-Pen-Eingang
Einfache 5 Volt-Betriebsspannung
TTL-kompatible Ein- und Ausgänge

Urspünglich wurde der 6845 von Motorola für den Einsatz in Computersystemen mit der Prozessor-Familie 68xx entwickelt. Auf Grund der außergewöhnlichen Flexibilität und einfachen Handhabung ist dieser Controller aber in sehr vielen Systemen finden. Selbst bei so leistungsstarken Systemen wie z.B. Sirius ist dies IC zu finden.

Vss 1	VSYNC
RES*	HSYNC
LPSTB	RA 0
MA 0	RA 1
MA 1	RA 2
MA 2	RA 3
MA 3	RA 4
MA 4	D 0
MA 5	D 1
MA 6	D 2
MA 7	D 3
MA 8	D 4
MA 9	D 5
MA 10	D 6
MA 11	D 7
MA 12	CS*
MA 13	RS
DISPTMG	E
CUDISP	R/W*
Vcc	CCLK

1.4.1.1 Pinout des CRTC HD 6845

1.4.1 Die Anschlüsse des CRTC

Die Bedeutung der 40 Anschlußbeine ist wie folgt:

MA0 - 13 :Memory Adress Lines
Über diese 14 Anschlüsse werden die Speicherplätze des Bildspeichers adressiert.

RA0 - 4 :Raster Adress Lines
Diese 5 Anschlüsse wählen aus dem Charactergenerator die derzeitige Rasterzeile des darzustellenden Zeichens aus.

D0 - 7 :Bidirectional Data Bus
Über diese Pins werden Informationen in den Controller geschrieben und aus ihm herausgelesen.

R/W* :Read/Write*
Dieses Signal bestimmt die Datenrichtung auf den Datenleitungen. Bei einem Low können Daten vom Prozessor in den CRTC geschrieben werden, bei High werden sie aus dem CRTC gelesen.

CS* :Chip Select*
Um Datentransfer mit dem 6845 zu ermöglichen, muß er adressiert werden. Dies geschieht durch ein Low am CS*-Eingang.

RS :Register Select
Dieses Signal wird benötigt, um zwischen Adress-Register und 18 Control-Registern zu selektieren. Bei Lowpegel an RS kann auf das Adress-Register zugegriffen werden, bei einem High besteht Zugriff auf die Control-Register.

EN :Enable
Mit der steigenden Flanke dieses Signals werden die am IC anliegenden Prozessorsignale vom Controller übernommen.

RES* :Reset*

Ein Low an diesem Eingang setzt alle Zähler im CRTC zurück und alle Ausgänge werden Low.
Diese Funktion wird aber nur ausgeführt, wenn gleichzeitig der LPSTB-Eingang Low ist. Der Reset löscht nicht die Control-Register!

CLK :Character Clock

ist das Taktsignal, aus dem alle vom Monitor benötigten Signale durch Teilung abgeleitet werden.

HSYNC :Horizontal Sync

liefert das Signal für die horizontale Synchronisation des Monitors. Falsch eingestellter oder fehlender HSYNC äußert sich im 'Durchlaufen' des Bildes.

VSYNC :Vertical Sync

liefert das vom Monitor benötigte Signal zur vertikalen Synchronisation.

DISPTMG :Display Timing

Dieses Signal ist zu den Zeiten High, wenn das dem Monitor zugeführte Signal auf dem Bildschirm darzustellen ist. Mit Hilfe dieses Signals lassen sich die Strahlrückläufe unterdrücken.

CUREN :Cursor Enable

(oft auch als Cursor Display, CURDISP, bezeichnet) wird verwendet, wenn der Cursor nicht durch die Software, sondern den CRTC selbst gesteuert wird. Auch das Cursorblinken kann mit diesem Anschluß gesteuert werden.

LPSTB :Light Pen Strobe

Wird an diesem Eingang eine Low-High-Flanke angelegt, dann wird der derzeitige Zustand der MA-Leitungen in die Light-Pen-Register übertragen und gespeichert. Diese Register können ausgelesen

und in einem entsprechenden Programm verwendet werden.

1.4.2 Die internen Register des Video-Controllers

Wie bereits erwähnt, enthält der 6845 ein Adress-Register und 18 Control-Register. Da mit dem Signal RS, Register Select, aber nur zwischen zwei Adressen ausgewählt werden kann, stellt sich die Frage, wie alle 18 Control-Register über nur eine Adresse angesprochen werden können.
Die Lösung des Problems ist das Adress-Register. In das Adress-Register wird die Nummer des Control-Registers geschrieben, auf das man als nächstes zugreifen möchte. Dieses Verfahren mutet zwar etwas umständlich an, hat aber einen unbestreitbaren Vorteil. Auf diese Art belegt der CRTC eben nur zwei und nicht 18 oder gar 32 Adressen. Da außerdem der CRTC normalerweise nur einmal beim Einschalten des Gerätes programmiert wird, kann auch der Mehraufwand an Programmierung in Kauf genommen werden.

Aber betrachten wir nun die 18 Register etwas detaillierter. Die folgende Beschreibung fällt allerdings wegen der komplexen Struktur einzelner Register etwas trocken und schwer verständlich aus. Auch sind zum Verständnis einiger Register grundlegende Kenntnisse der Videotechnik nötig. Sollten Sie beim Lesen nicht alles verstehen, so trösten Sie sich mit der Gewißheit, daß der Videocontroller in Ihrem Computer nicht 'von Hand' programmiert werden muß.

In der folgenden Aufstellung bedeutet ein R hinter der Registerbezeichnung, daß dieses Register zu lesen ist, ein W bedeutet die Möglichkeit, dieses Register zu beschreiben. Beachten Sie, daß einige Register nur zu beschreiben oder zu lesen sind (gekennzeichnet durch -).

AR -/W Adress Register

Dieses 5-Bit-Register wird mit der Nummer des gewünschten Control-Registers geladen. Registerwerte 18 bis 31 werden ignoriert, die gültigen

Werte lauten 0 bis 17. Dieses Register wird angesprochen, wenn sowohl CS als auch RS Low sind.

R0 -/W Horizontal Total

In dieses 8-Bit-Register wird die Anzahl der Zeichen pro totaler Zeile eingetragen. Allerdings ist eine totale Zeile wesentlich länger als die am Bildschirm sichtbaren Zeichen, da auch die Zeiten für den Rand und den Strahlrücklauf mitgerechnet werden müssen. Entsprechend wird dieser Wert etwa 1.5 mal so groß wie die Anzahl der dargestellten Zeichen gewählt sein.

R1 -/W Horizontal Displayed

Dieses Register enthält die Anzahl der am Bildschirm darzustellenden Zeichen. Der hier eingetragene Wert muß kleiner als der von R0 sein.

R2 -/W Horizontal Sync Position

Der 8-Bit-Wert dieses Registers bestimmt den Zeitpunkt des HSync-Impulses. Wird der Wert von R2 verringert, so verschiebt sich das Monitorbild nach rechts, eine Erhöhung schiebt das Bild nach links.

R3 -/W Sync Width

Mit den unteren 4 Bit dieses Registers wird die Breite der HSync und VSync-Impulse festgelegt. Die oberen 4 Bit dieses Registers werden nicht benutzt.

R4 -/W Vertical Total

Die unteren 7 Bit dieses Registers bestimmen die Anzahl aller Rasterzeilen pro Bild. Der Wert bestimmt damit auch, ob eine Bildwiederholfrequenz von 50 oder 60 Hertz gewählt wird.

R5 -/W Vertical Total Adjust

Mit Hilfe der unteren 6 Bit dieses Registers kann ein Feinabgleich der Bildwiederholfrequenz vorgenommen werden.

R6 -/W Vertical Displayed

Die unteren 7 Bit dieses Registers bestimmen die Anzahl der tatsächlich dargestellten Rasterzeilen auf dem Monitor. Hier kann theoretisch jeder Wert programmiert werden, der kleiner als (R4) ist.

R7 -/W Vertical Sync Position

Der 7-Bit-Wert dieses Registers bestimmt den Zeitpunkt des VSync-Impulses. Wird der Wert von R7 verringert, so verschiebt sich das Monitorbild nach unten, eine Erhöhung schiebt das Bild nach oben.

R8 -/W Interlace

Mit den unteren beiden Bits dieses Registers wird bestimmt, ob die Darstellung mit oder ohne Zeilensprung-Verfahren (Interlace) erfolgen soll.

R9 -/W Maximum Raster Adress

Dieses 5-Bit-Register bestimmt die Anzahl der Rasterzeilen der darzustellenden Zeichen.

R10 -/W Cursor Start Raster

Die Bits 0 bis 4 dieses Registers bestimmen, auf welcher Rasterzeile der Cursor beginnen soll. Die Bits 5 und 6 legen den Cursormodus fest. Der Cursormodus wird dabei mit den Bits wie folgt festgelegt:

Bits 6	5	
0	0	Cursor nicht blinkend
0	1	Cursor nicht dargestellt
1	0	Cursor blinkt (ca 3 x pro Sek. / 50Hz)
1	1	Cursor blinkt (ca 1.5 x pro Sek. / 50 Hz)

R11 -/W Cursor End Raster

Entsprechend zu (R10) legen die unteren 5 Bit dieses Registers fest, auf welcher Rasterzeile der Cursor endet.

R12 R/W Start Adress High
Die Bits 0 bis 5 legen fest, ab welcher Adresse im gesamten 16k-Adressbereich des CRTC der Bildspeicher beginnt. Wird dieses Register gelesen, so sind die Bits 6 und 7 immer Low.

R13 R/W Start Adress Low
Dieses Register legt analog zu (R12) das niederwertige Adressbyte des zu adressierenden Bildschirmspeichers fest.

R14 R/W Cursor High
Die Bits 0 bis 5 dieses Registers stellen das High-Byte der momentanen Cursorposition dar.

R15 R/W Cursor Low
Analog zu (R14) wird in diesem Register das Low-Byte der Cursor-Adresse abgelegt.
Da sowohl R14 als auch R 15 beschrieben und gelesen werden können, kann über diese Register die Cursorposition frei bestimmt werden.

R16 R/-
Dieses Register enthält nach einem positiven Strobeimpuls das Highbyte der zum Zeitpunkt des Impulses aktiven Bildschirmspeicheradresse. Die Bits 6 und 7 dieses Reisters sind immer Low.

R17 R/-
Analog zu R16 enthält dieses Register das Lowbyte zum Zeitpunkt des Light-Pen-Strobes.
Sowohl R16 wie auch R17 können nur gelesen werden.

1.5 Das RAM des CPC

Der im CPC eingebaute Schreib/Lesespeicher (RAM) wird nicht nur als Daten- und Programmspeicher eingesetzt. Auch die Bildschirminformation wird in diesem Speicher untergebracht.

Nachdem in den vorherigen Kapiteln die drei wichtigsten Bausteine des CPC, der Prozessor, das Gate Array und der Video-Controller, detailliert besprochen wurden, werfen wir in diesem Abschnitt einen Blick auf das Zusammenspiel dieser drei Komponenten beim Zugriff auf die Speicher-ICs. Dabei wird auch geklärt, wie der Video-Controller das RAM anspricht, um Zeichen auf dem Bildschirm darzustellen. Auch die Adressierung der im CPC 6128 eingebauten zusätzlichen 64 KByte werden wir ausführlich besprechen.

Zuvor aber wollen wir einen kleinen Abstecher machen und uns anschauen, wie die verwendeten dynamischen RAM-Bausteine mit ihren 8 Adress-Pins überhaupt funktionieren.

Als erstes müssen wir klären, wie die Adressierung von 65536 Speicherzellen mit den zur Verfügung stehenden 8 Adress-Anschlüssen möglich ist.
Das grundsätzliche Funktionsprinzip besteht darin, die 16-Bit-Adresse in zwei Hälften zu teilen und diese beiden Adress-Bytes nacheinander an die Adress-Pins der RAMs zu legen. Dieser Vorgang wird Multiplexen genannt. Allerdings erfordert das Multiplexen Steuersignale, die den RAMs mitteilen, welche Information zur Zeit an den Adressanschlüssen anliegt.

An diesem Punkt kommen die vom Gate Array gelieferten Signale RAS* und CAS* ins Spiel. Nachdem ein Adress-Byte an den RAMs anliegt, wird ihnen durch einen High-Low-Wechsel des Signals RAS* mitgeteilt, daß eine Adresshälfte parat ist. Mit der negativen Flanke (dem High-Low Wechsel) des RAS* wird die anliegende Adressinformation in den RAMs gespeichert.
Jetzt kann die zweite Hälfte der Adresse an das RAM angelegt

werden. Sobald dieses Adressbyte anliegt, wird das CAS*-Signal Low. Damit hat das RAM die gesamte 16-Bit-Adresse erhalten und wählt die gewünschte Speicherzelle an. Diese Zelle kann jetzt beschrieben oder ausgelesen werden.
Die Umschaltung der Adresshälften muß natürlich auch von einem passenden Signal übernommen werden, im CPC ist es das Signal CAS-ADDR*.
Als Umschalter oder Multiplexer (meint beides dasselbe, Multiplexer hört sich nur viel fachmännischer an) arbeiten vier ICs vom Typ 74LS153. Die Funktion dieser ICs kann man sich am besten wie zwei elektronisch gesteuerte Drehschalter vorstellen. Jeder der beiden Schalter hat vier Eingangsanschlüsse und einen Ausgang. Über zwei Steuereingänge kann entschieden werden, welcher der vier Eingänge mit dem Ausgang verbunden ist.

Die beiden Steuereingänge werden von den Signalen CPU-ADDR* und CAS-ADDR* angesteuert. Mit dem Signal CPU-ADDR* wird entschieden, ob der Prozessor oder der CRTC eine Adresse an das RAM legen kann, CAS-ADDR* schaltet zwischen den jeweiligen Adresshälften um.

Diese genaue Zuordnung der Adress-Anschlüsse von Prozessor und Video-Controller zeigt die folgende Tabelle.

Z80	6845	Z80	6845
A0	CCLK	A8	MA7
A1	MA0	A9	MA8
A2	MA1	A10	MA9
A3	MA2	A11	RA0
A4	MA3	A12	RA1
A5	MA4	A13	RA2
A6	MA5	NA14	MA12
A7	MA6	NA15	MA13

Wie man sieht, liegen alle Adressbits des Prozessors über die Multiplexer an den Adress-Anschlüssen der RAMs. Beim CPC6128 allerdings liegen die Adress-Signale A14 und A15

nicht direkt an den Multiplexern. Hier ist der Baustein zur Speicherumschaltung zwischengeschaltet. Aber auch der Video-Controller adressiert unter Zuhilfenahme des CCLK den gesamten 64K-Speicherbereich. Das aber steht im Gegensatz zum vorigen Kapitel, wo ja gesagt wurde, daß der CRTC einen Bereich von nur 16K adressieren kann.
Diese Aussage ist insoweit richtig, da als Adressleitungen nur die 14 mit MA (Memory Adress Line) bezeichneten Anschlüsse gezählt werden. Diese 14 Anschlüsse ermöglichen einen Adressbereich von 16 K.

Die im CPC eingesetzte Betriebsart des 6845 zur Adressierung des Video-Speichers wird nicht sehr häufig verwendet. Mit den Anschlüssen RA0 bis RA4 wird normalerweise ein fest programmiertes Zeichen- oder Character-Rom angesteuert, das die Bitmuster für die auf dem Bildschirm darzustellenden Zeichen enthält.
Üblicherweise haben Computer einen als Video-RAM bezeichneten Speicherbereich, in dem die auf dem Bildschirm darzustellenden Zeichen gespeichert werden. In diesem Speicher belegt jede Zeichenposition ein Byte. Das ergibt z.B. bei der Darstellung von 80 x 25 Zeichen einen Speicherbedarf von 2000 Bytes.
Nun kann aber in einem Byte nicht die gesamte zur Darstellung benötigte Information untergebracht werden. Jedes Zeichen besteht ja aus einer Anzahl von untereinander liegenden Punktereihen.
Auch beim CPC kann man diese Reihen auf dem Monitor erkennen. So besteht z.B. der Cursor aus 8 untereinander liegenden Reihen, in denen alle Bildpunkte 'an' sind. Bei der Darstellung von Buchstaben oder Ziffern sind nur bestimmte für die Darstellung des Zeichens erforderliche Punkte in einer Reihe an. Diese Punkte-Muster lassen sich durch Bitmuster speichern, wobei üblicherweise ein gesetztes Bit einem Punkt auf dem Bildschirm entspricht.
Die RA-Anschlüsse werden nun benötigt, um die einzelnen Reihen, also Bitmuster, aus dem Zeichen-Rom zu erhalten. Dazu werden die RA-Anschlüsse als Adressleitungen für das Zeichen-Rom verwendet.

Wie man sich vorstellen kann, ist es bei Verwendung von fest programmierten Zeichen-Roms nicht möglich, auf dem Bildschirm hochauflösende Grafik zu erzeugen. Nach diesem Prinzip konstruierte Computer sind an den eingebauten Zeichensatz gebunden.
Beim CPC entfällt aber dieses herkömmliche Character-Rom, hier hat man einen gänzlich anderen Weg beschritten.
Da die RA-Anschlüsse direkt den Speicher adressieren, muß also die Punkte-Information auch im RAM untergebracht sein. Nur durch diesen Schaltungstrick ist es möglich, jedes beliebige Bitmuster auf dem Monitor zu erzeugen, sprich Grafik in den bekannten Grenzen darzustellen.

Doch bevor wir uns dem konkreten Aufbau des Video-Speichers zuwenden, soll endlich das Signal CCLK erklärt werden. Dazu ist allerdings ein klein wenig Mathematik nötig.
Der CRTC wird mit einer Taktfrequenz von 1 MHZ angesteuert. Mit jedem Taktimpuls wird eine Speicherzelle adressiert. In dieser Zelle steht bitweise verschlüsselt die Information, welche Punkte auf dem Bildschirm 'an', also in der Schreibfarbe dargestellt sein sollen. Da eine Frequenz von 1 MHz einer Periodendauer von 1 Mikrosekunde entspricht, steht für die Darstellung jedes Punktes genau ein Achtel der Taktfrequenz zur Verfügung. Das ist eine Zeit von 0.125 Mikrosekunden. Um alle 640 Punkte einer Zeile darzustellen, ist somit eine Zeit von 80 Mikrosekunden erforderlich.
Da aber das die Dauer einer Zeile bestimmende V-Sync-Signal eine Periodendauer von 52 Mikrosekunden hat, kann diese Rechnung nicht aufgehen. Mit diesen Werten lassen sich maximal 40 Zeichen darstellen.
Ein Ausweg aus diesem Problem ist eine spezielle Betriebsart der RAMs, der Page Adress-Mode. Hat ein RAM nach dem Anlegen der RAS- und CAS-Signale den Inhalt der gewünschten Speicherzelle auf die Datenausgänge gelegt, dann reicht es, mit einem weiteren CAS-Impuls nur eine neue Adress-Hälfte an die RAMs zu legen, um das nächste Byte zu erhalten. Das setzt natürlich voraus, daß sich nur eine Hälfte der Adressinformation ändert.

Genau diese Eigenschaft haben die Entwickler des CPC genutzt. Natürlich muß die Adressinformation zu den beiden CAS-Impulsen unterschiedlich sein, sonst liest man dieselbe Speicherzelle zweimal. Das ist aber beim CCLK-Signal gegeben, es schaltet genau zwischen den beiden CAS-Impulsen um. Dieses Signal wird vom Multiplexer auf das Adressbit 0 (vom Prozessor aus gesehen) gelegt, wenn das Signal CAS-ADDR auf Low, das Signal CPU-ADDR dagegen auf High ist. Es stellt damit das unterste Adressbit des Video-RAMs dar.
Die beiden schnell aufeinander gelieferten Bytes aus dem Video-RAM werden im Gate Array zwischengespeichert, in die für den Monitor benötigte serielle Form umgewandelt und zusammen mit den Farbinformationen an den RGB-Ausgang geliefert.

Bleiben noch die beiden Signale MA12 und MA 13. Mit Hilfe dieser beiden Bits wird innerhalb von 16k-Schritten der Beginn des Video-RAMs bestimmt. Üblicherweise sind diese Bits gesetzt, das Video-RAM beginnt also bei &C000. Aber auch ein Video-Bereich von &4000 bis &7FFF ist bei entsprechender Programmierung möglich.

1.5.1 Die zusätzlichen 64 K des 6128

Die Klärung der Zusammenhänge der Speicherumschaltung im 6128 war etwas verzwickt. Mit einfachen PEEKs und POKEs war dem Problem nicht beizukommen. Als Ansatzpunkt blieb nur das mit dem Rechner ausgelieferte Programm 'BANKMAN'. Da dieses Programm jedoch aus unerfindlichen Gründen geschützt ist, wurde die Sache noch etwas erschwert. Wie dem auch sei, nach einiger Zeit des Probierens und Experimentierens können zumindest die wesentlichen Grundlagen der Speicherumschaltung beschrieben werden.

Bevor wir jedoch die Speicherumschaltung beschreiben, sollen zwei Begriffe geklärt werden. Mit Bank bezeichen wir einen Speicherbereich von 64 KByte, ein Block dagegen ist 16 KByte groß. Diese beiden Bezeichnungen werden im folgenden Abschnitt häufig benutzt werden.

Die Organisation des Speichers wird durch einen PAL-Baustein vom Typ HAL16L8 vorgenommen. An diesen Baustein sind die Datenleitungen D0 bis D2 sowie D6 und D7, die Adressansignale A14 und A14, das Signal IOWR*, das Signal CAS sowie RESET und CPU* angelegt. Als Ausgänge stehen die Signale NA14, NA15, CAS0 und CAS1 zur verfügung. Der PAL selbst belegt die Portadresse &H7Fxx, genau wie das Gate Array. Aus der Beschreibung es Gate Array wissen Sie, daß die Registerauswahl im GA durch den Zustand der Datenbits D6 und D7 vorgenommen wird. Die Kombination, bei der beide Datenbits Eins (High) sind, selektiert kein Register im GA. Statt dessen wird die Information in den Datenbits D0 bis D2 im PAL ausgewertet. Über diese Information wird die Speicherkonfiguration umgeschaltet.

Nach einem RESET-Impuls verhält sich der Rechner, als ob nur eine 64 KByte-Bank eingebaut ist. Die Adress-Signale A14 und A15 vom Z80 werden ohne Modifikation über den PAL an die Adress-Multipexer geführt. Das CAS-Signal des GA wird über den PAL auf den Pin CAS0 gelegt. Das CAS1-Signal ist vorläufig inaktiv. Damit wird die erste Bank im CPC bei Speicherzugriffen aktiviert. Der Refresh übrigens ist auch für die zweite Bank sichergestellt, da hierfür nur das RAS-Signal nötig ist. dieses Signal aber liegt parallel an beiden Banks.
Wenn jedoch ein entsprechender Wert an den PAL ausgegeben wird, so ändern sich die Speicherverhältnisse im CPC deutlich. Aber überlegen wir doch einmal, welche Werte überhaupt in Frage kommen. Die Portadresse war ja bereits klar. Weiterhin wissen wir, daß die Datenbits D6 und D7 gesetzt sein müssen, um nicht fälschlicherweise Register im GA anzusprechen. Die Datenbits D3 bis D5 werden nicht abgefragt, da sie nicht mit dem PAL verbunden sind. Damit sind die möglichen Werte klar. Es sind nur die Wert &C0 bis &C7 möglich. Was aber bewirken sie?

Leider ist es durch den Aufbau des CPC recht schwierig, alle Kombinationen zu klären. Teilweise wird nämlich annähernd der komplette Speicher umgeschaltet. Nach der Umschaltung

ist dann aber auch das Programm zur Umschaltung verschwunden. Die Folge ist ein klassischer Systemabsturz. Die für Sie wesentlichen Kombinationen können wir aber doch mitteilen. Bei diesen Werten wird der Speicher im Adressbereich von &4000 bis &7FFF der Bank 0 gegen einen Block der Bank 1 ausgetauscht. Die dafür benötigten Werte finden Sie in der folgenden Tabelle:

```
&C0   Bank0, Block1   Original-Zustand
&C4   Bank1, Block0
&C5   Bank1, Block1
&C6   Bank1, Block2
&C7   Bank1, Block3
```

Wir einer der Werte zwischen &C4 und &C7 auf die Portadresse &7Fxx ausgegeben, wird im Bereich &4000 bis &7FFF das CAS0-Signal inaktiv. Statt dessen wird das CAS1-Signal aktiv. Auch wird die Information auf den Adress-Pins A14 und A15 des Z80 durch den PAL modifiziert. Ausnahme ist hierbei der Wert &C5, der den selben Adressbereich der zweiten Bank anspricht. Bie &C4 z.B. wird jedoch der Adressbereich &0000 bis &3FFF der zweiten Bank adressiert, ohne daß der Prozessor etwas davon 'merkt'. Für ihn befindet sich das RAM weiterhin im von ihm gewünschten ADressbereich. Entsprechend gilt für den Wert &C6 der Adressbereich von &8000 bis &BFFF, für &C7 der Bereich von &C000 bis &FFFF der zweiten Bank.

Leider konnte die genaue Bedeutung der Werte &C1 bis &C3 nicht geklärt werden. Diese Werte spielen sicher eine große Rolle beim CP/M 3.0, da mit den anderen Werten keine TPA von 61 KByte realisierbar ist. Allerdings können wir für interessierte Leser die Speicherbelegung unter CP/M 3.0 mitteilen.
Unter diesem Betriebssystem muß man mit drei parallel liegenden RAM-Bereichen 'hantieren'. Natürlich müssen Sie nicht selber irgendwelche Speicherumschaltungen vornehmen. Das erledigt das CP/M ja für Sie.
In allen drei Banks ist der Adressbereich von &C000 bis &FFFF identisch. Dieser Bereich wird selber nie

umgeschaltet, da über ihn die Umschaltung der anderen Bereiche vorgenommen wird. In diesem Adressbereich ist das obere Ende der TPA, sowie die immer residenten Teile des BIOS und BDOS angesiedelt.
In der Bank 0 finden sich aber noch drei andere Blöcke. Der erste Block (&0000 bis &3FFF) enthält den unteren Jump-Block, der bereits nach dem Einschalten des CPC aus dem ROM in das untere RAM kopiert wird. Im Block 1 der Bank 0 (&4000 bis &7FFF) befindet sich das Bildschirm-RAM. Im Block 2 schließlich befindet sich der größte Teil des BIOS und des BDOS sowie die notwendigen Jump-Blocks, die ja unter CP/M 2.2 einer Erweiterung der TPA im Wege waren.
Die Bank 2 besteht aus den Blöcken 0 bis 2 und enthält den größten Teil der TPA. Der noch fehlende Teil der TPA ist im Block 3 dieser Bank enthalten. Dieser Block ist aber ja für alle drei Banks gleich.
In der Bank 2 schließlich ist noch einmal der Bereich von &4000 bis &7FFF belegt. In diesem Bereich sind der CCP und die vom CP/M benötigten Hash-Tabellen untergebracht.

Wenn Sie mit den in diesem Kapitel nicht erklärten Werten zur Speicherumschaltung experimentieren wollen, so sollten Sie die folgenden Tips beherzigen. Zunächst sollten Sie den Bildschirmspeicher von &C000 nach &4000 verlegen. Danach sollten Sie Ihr Testprogramm in den freigewordenen Bereich bei &C000 legen und starten. Am besten wäre natürlich ein kleines Monitor-Programm, das in diesem Bereich liegt. Der Grund dafür liegt in der Tatsache, daß dieser Bereich wahrscheinlich nie weggeschaltet wird, wie dies bei den anderen Bereiche passieren kann. Das Monitor-Programm muß aber vor jedem Aufruf von System-Routinen über die Jump-Blocks die Speicherkonfiguration wieder in den Original-Zustand versetzen, also den Wert &C0 auf die entsprechende Portadresse ausgeben. Wenn Sie dieses Rückschalten vergessen, kann es passieren, daß durch Ihre Umschaltung die Jump-Blocks einfach nicht vorhanden sind. Dann fällt Ihr Rechner auf die Nase.

1.6 Das Video-Ram zwischen Z80 und 6845

Probieren Sie am CPC doch einmal dieses kurze Programm:

```
10 MODE 2
20 FOR i = &c000 TO &ffff
30 POKE i,255
40 NEXT i
```

Sie erhalten auf dem Bildschirm eine dünne Linie, die von der linken oberen Ecke schnell nach rechts gezeichnet wird. Am Ende der ersten Linie wird sie genau 8 Reihen tiefer fortgesetzt.
Ist der Bildschirm mit diesen dünnen Linien einmal gefüllt, so beginnt das Ganze wieder links oben, diesmal aber eine Punktereihe tiefer.
Probieren Sie das Programm auch einmal im MODE 1 und MODE 0.

Danach ändern Sie versuchsweise die Zeile 30 in:

```
30 POKE i,1
```

Jetzt erhalten wir eine Punktereihe, die den Bildschirm zu senkrechten Reihen füllt.

Wenn das Programm im Mode 2 lief, dann sieht man, daß die senkrechten Reihen an der rechten Seite der Zeichen stehen. Im Mode 1 erhalten wir zwei senkrechte Reihen pro Charakter, im Mode 0 sind es sogar 4.

Wir wollen eine letzte Änderung am Programm vornehmen. Löschen Sie dazu die Zeile 10 des Programms und geben Sie 'MODE 2' im Direktmodus ein. Der Bildschirm wird gelöscht und 'READY' erscheint in der linken oberen Ecke. Betätigen Sie die Cursor-Down Taste (Pfeil nach unten) bis die Ready-Meldung aus dem Bild verschwindet. Der Cursor steht jetzt auf der letzten Eingabezeile. Lassen Sie das Programm noch einmal laufen.

Das Ergebnis ist einigermaßen irritierend.

Dieses kleine Programm hat uns gleich mehrere wichtige Dinge verraten.
Zum einen haben wir damit bewiesen, daß der Bildschirmspeicher bei &C000 beginnt und bei &FFFF aufhört. Überraschenderweise ist Lage und Größe der Bildschirmspeicher in allen drei Modi gleich. Es wird also nicht zwischen Modus 0 und Modus 2 unterschieden. Nur die erzeugten Farben sind unterschiedlich.

Allerdings gibt ein 16k-Bytes großer Bildschirmspeicher im Mode 0, also bei 20 Zeichen pro Zeile offensichtlich wenig Sinn. 20 Zeichen mal 25 Zeilen ergibt nur 500 Zeichen auf dem Bildschirm. Warum benötigt der CPC scheinbar 16384 Speicherplätze, um diese 500 Zeichen darzustellen?

Die Antwort ist recht einfach. Wie bereits erwähnt besitzt der CPC keinen Video-RAM, in dem ein Zeichen in einem Byte gespeichert wird.
Im 80-Zeichenmodus belegt ein Zeichen auf dem Bildschirm 8 Bytes, bei 40 Zeichen sind es 16 Bytes und im 20-Zeichen-Modus ganze 32 Bytes. Das läßt sich auch aus dem Programm ersehen, welches die senkrechten Linien erzeugte.
Unsere Darstellung 1.6.0.1 macht den Aufbau eines Zeichens noch einmal deutlich. Dabei soll das Zeichen im Mode 2 in der linken oberen Bildschirmecke stehen.

Der 80-Zeichen-Modus ist in dieser Hinsicht am einfachsten zu verstehen, da ein gesetztes Bit einen Punkt in der aktuellen Zeichen- (Pen-) Farbe erzeugt. Ist ein Bit dagegen nicht gesetzt, so erscheint an dieser Stelle auf dem Bildschirm die Hintergrundfarbe. Da im Mode 2 nur eine Zeichenfarbe möglich ist, gibt es keine weiteren Möglichkeiten.

Wofür werden aber im Mode 0 32 Bytes für ein Zeichen benötigt?

Diese Zusammenhänge sind bei den Modi 0 und 1 nicht mehr so einfach zu beschreiben. Sie sollten das folgende kleine Programm einmal eintippen und die angezeigten Ergebnisse bei der Lektüre vor Augen haben. Dadurch werden die Beschreibungen sicher verständlicher, als wenn Sie einen reinen 'Trockenkurs' versuchen.

```
10 MODE 2
20 REM
30 PRINT "A"
40 FOR adress = &C000 TO &F800 STEP &800
50  p$ = BIN$(PEEK(adress),8)
60  FOR l = 1 TO 8
70   IF MID$(p$,l,1) = "1" THEN PRINT "X"; ELSE PRINT ".";
80  NEXT l
90  PRINT
100 NEXT adress
```

Lassen Sie dieses Programm so wie beschrieben laufen, dann erhalten Sie ein Bild, das der abgedruckten Matrix des 'A' gleicht.

Ändern Sie nun einmal den Mode-Befehl in der Zeile 10 in 'MODE 1' und lassen Sie das Programm laufen. Das Ergebnis ist einigermaßen verblüffend.
Daß sich nur die halbe Matrix in den ausgelesenen Bytes befindet, war anzunehmen. Daß aber diese Matrix auch nur ein halbes Byte, also die Bits 4 bis 7, beansprucht, verwirrt zunächst.

Wir kommen der Klärung des Rätsels aber näher, wenn Sie die Zeile 20 ersetzen:

```
20 PEN 2
```

Außer der geänderten Schreib-(PEN-)farbe hat sich auch das durch unser Programm angezeigte Bitmuster geändert. Das aber ist die Lösung unseres Problems!

Wenn Sie mit dem CPC bereits etwas vertraut sind, werden Sie wissen, daß im 40-Zeichen-Modus 4 Farben möglich sind. Diese vier Farben lassen sich einfach mit dem Zeichen selbst abspeichern, in dem nur vier Bit für die gesetzten Pixel maßgeblich sind, und Low- und High-Nibble (ein Nibble = ein Halb-Byte, 4 Bit) über die Farben entscheiden. Bei dem verwendeten Prinzip muß nur das Gate Array die Pixel für die Anzeige in horizontaler Richtung verdoppeln, um auch tatsächlich 8 Punkte darzustellen, wo nur vier Punkte gespeichert sind.

Im Mode 0 bei der Darstellung von 20 Zeichen pro Zeile wird diese Methode noch erweitert. Hier sind es nur zwei Bit, welche die Pixel-Information enthalten. Die Stellung der zwei Pixel innerhalb des Bytes bestimmen die Farbe, in der dieses Pixel dargestellt werden soll. Damit sind insgesamt 16 Kombinationen möglich, genau die Anzahl der zur Verfügung stehenden Farben. Da nur zwei Pixel in einem Byte gespeichert sind, werden 4 Byte für eine Pixel-Zeile benötigt, insgesamt also 8 x 4 = 32 Bytes für ein Zeichen in 16 verschiedenen möglichen Farben.

Probieren Sie doch einfach das Programm im Modus 0 mit verschiedenen Werten für das PEN-Kommando aus. Sie werden dann schnell hinter das Funktionsprinzip kommen.

Damit sind die beiden ersten Punkte vom Beginn des Kapitels geklärt. Unklar dagegen ist noch der Punkt der 'Verschiebung' des Bildschirmrams. Dieses Problem ist in der Hardware des CPC begründet.

Auch ein Z80 mit einer Taktfrequenz von 4 MHz benötigt zum Verschieben eines 16K-Datenblocks einige Zeit. Um z.B. beim Listen eines längeren Basicprogramms nicht für jede neue Zeile den gesamten Video-Ram-Bereich um 640 Speicher-Plätze zu verschieben, hat man eine spezielle Eigenschaft des CRTC genutzt. Durch entsprechende Programmierung der Register 12 und 13 des 6845 kann der Bildschirm praktisch auf jeder geraden Speicherzelle des Video-Rams beginnen. Dadurch kann das Scrollen sehr viel schneller passieren, da nur die

entsprechenden Register mit den nötigen Werten versorgt werden müssen. Die neue Zeile am unteren Bildrand ist schnell gelöscht und mit den Zeichen versehen.

Ein Start des Video-Ram auf einer ungeraden Adresse, also z.B. bei &C001 ist wegen der beschriebenen Verwendung des Signals CCLK als Adressbit nicht möglich.

Das folgende Programm zeigt, daß eine Manipulation der genannten Register auch von Basic aus zu bewerkstelligen ist:

```
10 adrreg = &bc00       : REM Adressregister des 6845
20 datreg = &bd00       : REM Port des Datenregisters
30 OUT adrreg,13        : REM Register wählen
40 FOR offset = 1 TO 40
50 OUT datreg,offset    : REM 40 mal ändern
60 FOR warten = 1 TO 40 : REM und etwas warten
70 NEXT warten,offset
```

In diesem Programm wird der Bildschirminhalt horizontal gescrollt. Ohne die Warteschleife würde das Scrollen so schnell ablaufen, daß man den Vorgang mit dem Auge gar nicht verfolgen könnte.

Auch vertikales Scrollen läßt sich von Basic aus programmieren. Allerdings müssen dann beide Register, Low- und Highbyte, manipuliert werden. Da aber zwischen den beiden OUT-Befehlen recht viel Zeit vergeht, kommt es zu unangenehmen Flimmer-Erscheinungen.

Es gibt beim Video-Ram aber noch eine Besonderheit zu beachten.

Rechnen wir die bekannten Werte einmal zusammen.
Im Mode 2 besteht ein Zeichen aus 8 Bytes. In einer Zeile haben 80 Zeichen Platz und es sind 25 Zeilen auf dem Bildschirm möglich. Das ergibt einen gesamten Speicherplatzbedarf von 80 x 25 x 8 = 16000 Bytes. Ein

16K-Speicherbereich hat aber 2 hoch 14 = 16384 Speicherplätze. Wo sind die fehlenden 384 Bytes?
Ganz einfach. Sie werden nicht benötigt. Jedenfalls nicht, so lange der Bildschirm nicht gescrollt wird.
Hier könnten kurzfristig zu speichernde Werte untergebracht werden, die aber spätestens beim nächsten CLS mit Sicherheit verschwunden sind.

Ein komplettes Diagramm des Bildschirmspeichers könnte etwa wie auf unserer Grafik 1.6.0.2 dargestellt werden. Dabei wird davon ausgegangen, daß der Bildschirm tatsächlich mit dem Speicherplatz &C000 in der linken oberen Ecke beginnt. Dieser Zustand ist nach jedem MODE-Kommando gegeben.

Die Adressen &C7D0-&C7FF, &CFD0-&CFFF, ..., &FFD0-&FFFF werden in dieser Konfiguration nicht verwendet.

Sie werden sich jetzt sicher fragen, wie um alles in der Welt mit dieser verrückten Organisation des Bildschirmspeichers jemals vernünftig Grafik programmiert werden kann. Auch scheint es fast unmöglich, ein Zeichen vom Bildschirm zu lesen. Bei anderen Rechnern ist das kein Problem, da kann mit einem POKE ein Zeichen auf dem Bildschirm plaziert werden. Entsprechend kann der Inhalt des Video-Ram mit PEEK ausgelesen werden.
Weiterhin ist üblicherweise sicher, daß das Video-Ram auf einer bestimmten Adresse anfängt.

Nun ist aber nicht alles so schlimm, wie es auf den ersten Blick erscheint. Das Betriebssystem ist ja auch in der Lage, mit den wechselnden Startadressen klarzukommen, oder z.B. ein Zeichen aus der Bildschirm-Matrix zu bestimmen, wie das bei jeder Benutzung der Copy-Taste passiert. Die dafür benötigten Routinen können auch von selbsterstellten Maschinenprogrammen genutzt werden.
Viele dieser Routinen des Betriebssystems finden Sie in einem späteren Kapitel. Konkret zeigen wir die Nutzung der Grafik in einem Beispiel zum Zeichnen von Rechtecken und in einem Programm zum Erzeugen einer Grafik-Hardcopy.

1.7 Der Parallel-Schnittstellenbaustein 8255

Ursprünglich von INTEL für den 8080 entwickelt, eignet sich der 8255 als programmierbarer Mehrzweck-I/O-Baustein (I/O = Input/Output, Ein/Ausgabe) auch für andere Prozessoren. Der 8255 verfügt über insgesamt 24 Leitungen, über die Signale aus- oder eingegeben werden können. Jeweils 8 Leitungen bilden einen 8-Bit-Port, wobei der dritte Port in zwei getrennt programmierbare Hälften geteilt werden kann.

Die wichtigsten Leistungsmerkmale des 8255 sind:

24 programmierbare I/O-Anschlüsse.
Einfache Betriebsspannung 5 Volt.
Vollständig TTL-Kompatibel.
Drei leistungsfähige Betriebsarten programmierbar.
Jeder Port getrennt programmierbar.
Hoher Ausgangsstrom 1 mA bei 1.5 Volt Spannung.
Funktion Bit setzen/Bit Rücksetzen möglich.

1.7.1 Die Anschlußbelegung des 8255

Die Pinbelegung des 8255 ist im unten stehenden Bild gezeigt. Es bedeuten:

D0 - D7 : Data Lines

Diese Anschlüsse werden mit dem Datenbus des Prozessors verbunden. Sie dienen dem Transfer der Daten vom und zum Prozessor.

CS : Chip Select

Durch ein Low an diesem Anschluß wird der Baustein ausgewählt. Die jetzt an den RD-, WR- und Data-Leitungen anliegenden Signale werden vom 8255 akzeptiert.

RD : Read

Ein Low an diesem Anschluß veranlaßt den 8255 Daten oder Zustandsinformationen über den Datenbus an den Prozessor zu senden.

PA 3	1		PA 4
PA 2			PA 5
PA 1			PA 6
PA 0			PA 7
RD*			WR*
CS*			RESET
GND			D 0
A 1			D 1
A 0			D 2
PC 7			D 3
PC 6			D 4
PC 5			D 5
PC 4			D 6
PC 0			D 7
PC 1			Vcc
PC 2			PB 7
PC 3			PB 6
PB 0			PB 5
PB 1			PB 4
PB 2			PB 3

1.7.1.1 Pinout des Parallelport 8255

WR : Write

wird Low, wenn der Prozessor Daten oder Steuerbefehle an den 8255 schicken will.

A0, A1 : Adress Lines 0, 1

Über diese Anschlüsse wird zwischen den drei Datenkanälen und dem Steuer-Register ausgewählt. Häufig werden diese Anschlüsse mit den unteren beiden Adressleitungen des Prozessors verbunden.

RESET :

Ein High an diesem Eingang setzt alle Register einschließlich des Steuerregisters zurück. Die Portleitungen werden in die Betriebsart Eingabe gebracht.

PA0 - PA7 : Port A

Diese acht Leitungen stellen den I/O-Port A dar und können wahlweise als Eingang oder Ausgang verwendet werden.

PB0 - PB7 : Port B

Funktion wie Port A.

PC0 - PC7 : Port C

Funktion wie Port A.

1.7.2 Die Betriebsarten des 8255

Bevor auf die vier internen Register eingegangen wird, müssen wir zunächst die Möglichkeiten des ICs etwas genauer betrachten. Wie schon zu Beginn erläutert, verfügt der 8255 über drei mögliche Betriebsarten:

Betriebsart 0 : Einfache Ein/Ausgabe
Betriebsart 1 : Getastete Ein/Ausgabe
Betriebsart 2 : Zweiweg-Bus

Die Betriebsart 0 ist die einfachste und auch häufigste. In diesem Mode kann bestimmt werden, ob die Ports als Ausgabe- oder als Eingabe-Leitungen arbeiten sollen. Werden Leitungen als Ausgang programmiert und wird auf diese Ausgänge vom Prozessor eine Information gelegt, so wird der Wert gespeichert und die Ausgänge bleiben bis zur Neuprogrammierung oder einem Reset erhalten.
Als Eingang programmierte Ports liefern beim Lesen den momentanen Zustand an diesen Leitungen.

Sowohl Port A wie auch Port B lassen sich nur als ganzer Port für die gewünschte Datenrichtung programmieren. Es ist also nicht möglich, z.B. die Portbits PA0, PA3 und PA7 als Ausgang und die verbleibenden Anschlüsse als Eingang zu verwenden.
Allerdings kann der Port C in zwei Hälften geteilt werden. Die Datenrichtung jeder Hälfte kann getrennt programmiert werden.

Die Betriebsart 1 unterscheidet sich grundsätzlich vom Mode 0. In dieser Betriebsart ist ein Datentransfer mit Hand-Shake-Signalen in einer Richtung möglich. Jetzt spricht man nicht mehr von drei vorhandenen Ports, die beiden Hälften des Port C werden den anderen beiden Ports als Steuer- und Quittungssignale zur Verfügung gestellt. Man spricht dann von den beiden Gruppen A und B.
Die Gruppe A besteht aus Port A und den Bits 4-7 des Port C, die Gruppe B entsprechend aus dem Port B und den Bits 0-3 des Port C.
Um die Progammierung des Mode 1 komfortabel zu gestalten, besteht die Möglichkeit, jeweils ein spezielles Bit der entsprechenden Hälfte des Port B als Interrupt-Signal zu verwenden.
Ein solcher 8-Bit-Datentransfer wird z.B. bei Drucker-Schnittstellen verwendet. Hier zeigt ein Signal an, daß die Daten auf den Datenleitungen gültig sind. Ein rückgeführtes Signal meldet, ob der Empfänger, also in diesem Beispiel der Drucker, emfangsbereit ist oder ob die Daten korrekt emfangen wurden.

Diese Funktion kann vom 8255 wahlweise sowohl als Datenausgang wie auch als Eingang ausgeführt werden.

Die dritte Betriebsart (Mode 2) ist ein getasteter bidirektionaler Betrieb. Diese Funktion ist nur mit dem Port A möglich. Als Steuer- und Quittungssignale werden die Bits PC3-7 verwendet.
Ein möglicher Einsatz dieser Betriebsart wäre die Steuerung eines Floppy-Laufwerks, da hierbei die Daten ja sowohl zur Floppy wie auch von der Floppy zum Prozessor über dieselben Anschlüsse geführt werden müssen.

Zusätzlich besteht in allen drei Betriebsarten die Möglichkeit, die als Ausgang programmierten Bits des Port per Befehl gezielt zu setzen oder zu löschen.
Alle diese beschriebenen Betriebsarten lassen sich auch kombinieren. So ist es möglich, den Port A im Mode 0 als Ausgang, den Port B im Mode 1 als Eingang und die verbleibenden Bits des Port C als Eingang zu programmieren.

1.7.3 Steuerung des 8255, die Registerbeschreibung

Wenn man diese auf den ersten Blick verwirrende Anzahl der Möglichkeiten betrachtet, so fragt man sich unwillkürlich, wie alle Möglichkeiten und Kombinationen mit nur einem Steuerregister zu programmieren sind.
Der Trick, mit dem dies möglich wird, ist einfach. Das oberste Bit des Steuerworts wird als Kennzeichen-Bit verwendet. Ist dieses Bit im Steuerwort gesetzt, so haben die Bits 0 bis 6 die folgende Bedeutung:

```
Bit 0 : steuert Funktion des Port C Bits 0 bis 3
        1 = Eingang
        0 = Ausgang

Bit 1 : steuert Funktion des Port B
        1 = Eingang
        0 = Ausgang
```

```
Bit 2 : wählt den Mode der Gruppe B
        1 = Betriebsart 0
        0 = Betriebsart 1

Bit 3 : steuert Funktion des Port C Bits 4 bis 7
        1 = Eingang
        0 = Ausgang

Bit 4 : steuert Funktion des Port A
        1 = Eingang
        0 = Ausgang

Bit 6,5: wählen Modus der Gruppe A
        00 = Modus 0
        01 = Modus 1
        1x = Modus 2, Bit 5 ohne Bedeutung
```

Ist im Steuerwort das oberste Bit dagegen gelöscht, so wird über die Bits 0-3 die Funktion 'Bit setzen/Bit rücksetzen' des Port C definiert. Die Bedeutung dieser Bits lautet folgendermaßen:

```
Bit 0 : steuert Bit-Set/Bit-Reset
      1 = Bit setzen
      0 = Bit rücksetzen

Bits 3-1: Bitauswahl
      000 = PC0
      001 = PC1
      010 = PC2
      011 = PC3
      100 = PC4
      101 = PC5
      110 = PC6
      111 = PC7
```

Die Bits 4 bis 6 im Steuerwort sind bei gelöschtem siebtem Bit ohne Bedeutung.

Dieses Steuerregister kann nur beschrieben werden. Ein Lesen des Wertes ist nicht möglich. Wohl aber können die den Ports zugehörigen Register gelesen werden, auch wenn die Ports als Ausgang bestimmt sind. In diesem Fall entspricht der gelesene Wert dem Zustand der Portleitungen.

Der Zugriff auf die vier Register geschieht über die Anschluß-Pins A0 und A1. Die Zuordung zu den Registern zeigt die folgende Tabelle.

```
A1 A0
0 0 Port A Register
0 1 Port B Register
1 0 Port C Register
1 1 Steuerregister
```

1.7.4 Der Einsatz des 8255 im CPC

Nachdem wir uns einen Überblick über die vielfältigen Einsatzmöglichkeiten des 8255 geschaffen haben, wollen wir uns mit dem praktischen Betrieb dieses universellen Schnittstellen-Bausteins im CPC auseinandersetzen. Wie eigentlich fast alle ICs im CPC wird auch der 8255 optimal verwendet. Da bleibt kein Bit ungenutzt.

Doch werden wir konkret.
Der 8255 bedient die Tastatur, den Sound-Chip, den Motor des Cassetten-Recorders, erzeugt die Schreib-Signale des Recorders, liest den vom Recorder kommenden Bit-Strom, überprüft das V-Sync-Signal des CRTC, stellt fest, ob der Drucker empfangsbereit ist, fragt mit einem Bit den Zustand des EXP-Signals des Expansion Connectors ab, entscheidet über eine Brücke, ob die Erzeugung des Bildes nach PAL- oder SECAM-Norm mit 50 oder 60 Hertz Bildfrequenz erfolgen soll und zu guter Letzt bleiben noch ganze drei Bits über, die beim Einschalten Brücken abfragen und feststellen, was für einen Computer Sie sich gekauft haben. Der Zustand dieser Brücken entscheidet nämlich, ob Sie Schneider, Awa, Triumpf, Amstrad oder einen anderen der insgesamt acht verschiedenen

möglichen Firmennamen in der Einschaltmeldung auf den Blidschirm bekommen.

All diese Funktionen mit den zur Verfügung stehenden 24 I/O-Leitungen zu realisieren, zeugt von der ausgesprochenen Sparsamkeit und Pfiffigkeit der Hardware-Entwickler.

Der Datenbus ist direkt mit dem Datenbus des Prozessors verbunden. Das CS-Signal (Chip-Select) wird vom Adressbit A11 des Prozessors erzeugt. Die zur Registerauswahl vorhandenen Pins A0 und A1 des 8255 sind mit den Prozessoradresspins A8 und A9 verbunden.
Wie bereits erwähnt werden alle Peripherie-Bausteine im CPC über Port-Adressen angesprochen. Aus diesem Grund ist die Leitung RD* des 8255 mit dem Signal IORD* verbunden.
Erzeugt wird dieses Signal aus der Verknüpfung der Signale RD* und IORQ* des Z80. Nur wenn IORQ* und RD* Low sind, erscheint am Eingang RD* ein Low.
In ähnlicher Weise wird auch der WR*-Anschluß des 8255 angesteuert. Hier erscheint ein Low, wenn sowohl WR* als auch IORQ* des Z80 Low werden.

Aus diesen Daten können nun die Port-Adressen des 8255 bestimmt werden. Um z.B. in das Register 0, das Datenregister des Port A, einen Wert zu schreiben, müssen die Anschlüsse A11, A9 und A8 Low sein. In binärer Schreibweise erhalten wir für das Highbyte des Adressbusses den Wert:

A15	A14	A13	A12	A11	A10	A09	A08
1	1	1	1	0	1	0	0

Das entspricht dem hexadezimalen Wert &F4.

Die unteren 8 Adressbits gehen in die Auswahl des 8255 nicht ein, hier ist jeder Wert zwischen &00 und &FF möglich.
Auch die gesetzten Bits im Highbyte sind zur korrekten Adressierung des 8255 eigentlich nicht nötig, und so könnte man auf die Idee kommen, als High-Byte den Wert 00H

einzusetzen. Das würde sogar funktionieren. Da aber die Dekodierung der einzelnen Peripherie-ICs in ähnlich unvollständiger Weise vorgenommem wird, müssen die Bits gesetzt werden, sonst würden sich gleichzeitig andere ICs wie der CRTC oder das Gate Array mit angesprochen fühlen.

Doch zurück zu unserem Beispiel. Um also das Register A mit einem Wert zu laden, muß der Wert &F400 auf den Adressbus gelegt werden. Das kann mit den Befehlen

```
LD A,wert
LD BC,&F400
OUT (C),A
```

erreicht werden. Entsprechend kann z.B. das Port-Register C mit den Befehlen

```
LD BC,&F600
IN A,(C)
```

ausgelesen werden.

Grundsätzlich werden alle drei Ports im Modus 0 verwendet. Somit stehen alle 24 Anschlüsse als I/O-Leitungen zur Verfügung.

Der **Port A (&F400)** ist mit den 8 Datenleitungen des Sound Generators AY-3-8912 verbunden. Je nach geforderter Aktion wird der Port A als Ausgang oder Eingang programmiert.
Als Ausgang programmiert werden über die 8 Portleitungen die Steuerbefehle an den Sound-Chip geschickt. Diese Steuerbefehle finden Sie detailliert im Kapitel über die Programmierung des AY-3-8912. An dieser Stelle sei nur erwähnt, daß der Sound-Chip auch über einen bidirektionalen 8-Bit-Port verfügt. An diesen Port ist eine Seite der Tastatur-Matrix angeschlossen. Über den Port A des 8255 kann nun über den Umweg des Ports des AY-3-8912 festgestellt werden, ob eine Taste gedrückt ist. Zu diesem Zweck muß der Port A natürlich als Eingang programmiert werden.

Der **Port B (&F500)** wird fest als Eingangsport programmiert. Über diesen Port werden alle erwähnten Abfragen außer der Tastaturabfrage getätigt. Dabei sind die einzelnen Bits dieses Ports wie folgt belegt:

Bit 0 :

Dieses Bit fragt den Zustand des V-Sync des CRTC ab. Da diese Abfrage recht rasch gehen muß, kann durch einfaches Rotieren des mit INP gelesenen Wertes das Bit 0 ins Carry-Flag geschoben werden. So kann schnell der Zustand des V-Sync festgestellt werden.

Bit 1-3 :

Diese Bits entscheiden den Firmennamen in der Einschaltmeldung.

Bit 4 :

Dieses Bit ist mit einer Drahtbrücke verbunden. Ist die Brücke offen, so wird der Video-Controller für PAL-Betrieb mit 50 Hertz programmiert, eine geschlossene Brücke bewirkt eine Programmierung des CRTC für die SECAM-Norm mit 60 Hz Bildwiederholfrequenz. Diese unterschiedliche Programmierung ist wichtig, wenn der CPC über das Modul MP1 an einem Fernseher betrieben werden soll.

Bit 5 :

Dieses Bit fragt den Zustand des Signals EXP des Expansion Connectors ab.

Bit 6 :

Dieses Bit gibt den Zustand eines angeschlossenen Druckers wieder. Da der Drucker nicht zu allen Zeiten Zeichen empfangen kann, besteht die Möglichkeit, durch High-legen dieses Anschlusses einen Zeichentransfer zu unterbinden.

Bit 7 :

Über dieses Bit werden die vom Recorder mit TTL-Pegel gelieferten Daten eingelesen. Auch hier gilt das zu Bit 0 gesagte. Da diese Leitung sehr schnell geprüft werden muß, kann durch einmaliges Rotieren des Bit 7 in das Carry-Flag der Zustand dieser Leitung schnell bestimmt werden.

Der noch verbleibende **Port C (&F600)** ist im CPC fest als Ausgangsport programmiert. Mit vier seiner acht Leitungen steuert er einen Teil der Tastaturabfrage und zwei weitere Bits werden für den Recorder verwendet. Die restlichen beiden Bits werden für die Ansteuerung des Sound-Chip benötigt. Da die Leitungen des Port C direkt gesetzt und gelöscht werden können, eignet er sich natürlich besonders für diese Aufgaben.
Im einzelnen sind die Bits wie folgt verwendet:

Bit 0-3 :

Diese Bits steuern die Tastaturmatrix. Die als Ausgang programmierten vier Leitungen sind mit einem BCD-Dezimal-Decoder verbunden, der die binäre eine dezimale Information umwandelt. Dieser Decoder legt entsprechend der binären Eingangsinformation einen seiner zehn Ausgänge auf Masse. Dabei sind als Eingangskombinationen die Werte zwischen 0 und 9 erlaubt.

Bit 4 :

Dieses Bit steuert den Motor des Cassettenrecorders. Der Motor wird allerdings nicht direkt, sondern über einen Transistor (und ein nachgeschaltetes Relais) gesteuert. Liegt dieses Bit auf Masse, so wird der Motor ausgeschaltet.

Bit 5 :

Über diesen Pin des 8255 werden die Tonfrequenzen vom Computer geliefert, die auf dem Recorder aufgenommen werden sollen und beim Abhören diesen merkwürdigen Ton erzeugen.

Bit 6-7 :

Diese Portbits sind mit den Anschlüssen BC1 und BDIR des Sound-Chip verbunden und arbeiten als Chip-Select- und Strobe-Signal für den AY-3-8912. Eine detailliertere Beschreibung dieser Anschlüsse finden Sie im nächsten Kapitel über den Sound-Generator.

1.8 Der Sound Generator AY-3-8912

Der AY-3-8912 von General Instruments ist ein programmierbarer Sound Generator (PSG) der Spitzenklasse. Er wurde entwickelt für Telespiele, um diese mit besonders realistischem Sound zu versehen, nachdem die ersten Telespiele nur recht monotone Geräusche produzieren konnten. Um möglichst universell einsetzbar zu sein, wurde der PSG mit einer Vielzahl von Möglichkeiten zur Klangbeeinflussung versehen. Zusätzlich sagte man sich bei der Entwicklung dieses ICs wohl, daß in fast allen Einsatzgebieten auch irgendwelche Tasten, Joysticks oder Schalter abgefragt werden müssen. So gab man diesem PSG auch noch einen bidirektionalen 8-Bit-Parallelport mit.

Die Leistungsdaten dieses ICs im Überblick lauten:

Drei unabhängig programmierbare Ton-Oszillatoren
Ein programmierbarer Rausch-Generator
Vollständig software-gesteuerte Analogausgänge
Programmierbarer Mischer für Ton und Rauschen
15 logarithmisch gestufte Lautstärkestufen
Programmierbare Hüllkurven
Bidirektionaler 8-Bit-Datenport
TTL-Kompatibel
Einfache 5 Volt Betriebsspannung

Insgesamt verfügt der AY-3-8912 über 16 Register, von denen 15 Register genutzt werden können. Über diese Register können alle Klangmöglichkeiten des Chips programmiert werden.

Die Schaltung des PSG kann in einzelne Funktionsblöcke unterteilt werden.

Da ist zunächst der Block der Tongeneratoren. Die Tongeneratoren werden mit einem Taktsignal versorgt, das aus dem durch 16 geteilten Clock-Signal gewonnen wird. Die

CHANNEL C	1	DA 0
TEST 1		DA 1
Vcc		DA 2
CHANNEL B		DA 3
CHANNEL A		DA 4
Vss		DA 5
IOA 7		DA 6
IOA 6		DA 7
IOA 5		BC 1
IOA 4		BC 2
IOA 3		BDIR
IOA 2		A 8
IOA 1		RESET*
IOA 0		CLOCK

1.8.1.1 Soundchip AY–3–8912

Tongeneratoren sind zuständig für die grundsätzliche Erzeugung der drei rechteckförmigen Tonfrequenzen.

Der Rauschgenerator erzeugt ein frequenzmoduliertes Rechtecksignal, dessen Pulsbreite von einem Pseudo-Rauschgenerator beeinflußt wird.

Die Mischer koppeln die Ausgangssignale der drei Generatoren mit dem Rauschsignal. Die Kopplung kann für jeden Kanal getrennt programmiert werden.

Der Funktionsblock der Amplitudenkontrolle bietet dem Anwender zwei Möglichkeiten. Zum einen kann die Ausgangsamplitude (die Lautstärke) der drei Kanäle über die Programmierung des entsprechenden Lautstärke-Registers beeinflußt werden.
Alternativ besteht die Möglichkeit, sie vom PSG variabel zu beeinflussen. Dann wird der Ausgang des Hüllkurven-Registers genutzt, um die Lautstärke zu beeinflussen. Da die Hüllkurve (auch als Envelope, Umschlag oder Umhüllung bezeichnet) mit vier getrennten Parametern programmierbar ist, bestehen vielfältige Möglichkeiten der Tonbeeinflussung.

Der Funktionsblock der D/A-Wandler ist zuständig für die Erzeugung der Lautstärke der Ausgangssignale. Da die Lautstärke- und Envelope-Informationen als digitale Werte vorliegen, werden sie im D/A-Wandler umgesetzt.

Der letzte Funktionsblock hat mit der Tonerzeugung nichts zu tun. In diesem Block sind die zwei I/O-Ports untergebracht. Tatsächlich enthält der Chip des AY-3-8912 zwei vollständige I/O-Ports, von denen aber nur einer an Anschlußpins herausgeführt ist. Derselbe Chip wird im AY-3-8910 eingesetzt, bei dem beide Ports zur Verfügung stehen.

1.8.1 Die Anschlüsse des Sound Chip

Da die Bezeichnungen der Anschlüsse des PSG nicht unbedingt selbsterklärend sind, hier die detaillierte Beschreibung der Funktion der Pins:

DA0 - 7 :

Diese Anschlüsse des Sound Chips werden mit dem Datenbus des Prozessors verbunden. Die Bezeichnung DA deutet an, daß sowohl Daten als auch (Register-)Adressen über diese Anschlüsse geführt werden.

A8 :

Dieser Anschluß kann als ein CHIP-SELECT-Signal verstanden werden. Um Register des PSG anzusprechen, muß dieser Anschluß High sein.

BDIR & BC1,2 :

Der Anschluß BDIR-Signal (Bus DIRection) und die Anschlüsse BC1 und BC2 (Bus Control) steuern den Registerzugriff auf den PSG. Auf den ersten Blick mag die in der Tabelle gezeigte Zuordnung etwas seltsam erscheinen. Da dies IC jedoch ursprünglich als Baustein zum Prozessor 1610, einem speziellen 16-Bit-Prozessor von General Instruments, entwickelt wurde, hat man beim Entwurf auf die speziellen Eigenschaften und Steueranschlüsse dieses Prozessors Rücksicht genommen.

BDIR	BC2	BC1	Funktion des PSG
0	0	0	INACTIVE
0	0	1	LATCH ADRESS
0	1	0	INACTIVE
0	1	1	READ FROM PSG
1	0	0	LATCH ADRESS
1	0	1	INACTIVE
1	1	0	WRITE TO PSG
1	1	1	LATCH ADRESS

Innerhalb dieser Tabelle sind nur vier Kombinationen wirklich sinnvoll. Darum wird häufig der Anschluß BC2 auf +5 Volt gelegt. Die verbleibende Tabelle wird nur noch von den Signalen BDIR und BC1 bestimmt und sieht folgendermaßen aus:

BDIR	BC1	Funktion
0	0	PSG-Datenbus ist hochohmig
0	1	Daten aus PSG lesen
1	0	Daten in PSG schreiben
1	1	Registernummer in PSG schreiben

ANALOG A :

Dies ist der Ausgang des Kanals A. Hier können die von Kanal A erzeugten Töne abgenommen werden. Die maximale Ausgangsspannung ist 1 Vss.

ANALOG B :

Funktion wie Pin 1 für den Kanal B.

ANALOG C :

Funktion wie Pin 1 für den Kanal C.

IOA7 - 0 :

Die IOA-Anschlüsse stellen den 8-Bit-Port des PSG dar. Je nach Programmierung arbeiten die Anschlüsse als Ausgang oder Eingang. Dabei kann nur die Betriebsart für den ganzen Port eingestellt werden. Ein gemischter Betrieb (gleichzeitig Bits als Eingang, andere Bits als Ausgang) ist nicht möglich.

CLOCK :

Von der Frequenz dieses Signals werden alle Tonfrequenzen durch Teilung abgeleitet. Die Frequenz dieses Signals sollte zwischen 1 und 2 MHz liegen.

RESET :

Durch einen Low-Pegel an diesem Anschluß werden alle internen Register zurückgesetzt. Ohne Reset stehen nach dem Einschalten zufällige Werte in den verschiedenen Registern, die Folge wäre ein (wahrscheinlich) sehr unmusikalisches 'Geräusch'.

TEST1 :

Test1 wird nur von der Herstellerfirma verwendet und muß im Betrieb unbeschaltet bleiben.

Vcc :

An diesen Anschluß wird die Betriebsspannung von +5 Volt angelegt.

Vss :

Dies ist der Masse-Anschluß des PSG.

1.8.2 Die Funktion der einzelnen Register des 8912

Da jetzt geklärt ist, wie über die Anschlüsse BDIR und BC1 die Register grundsätzlich angesprochen werden können, wollen wir sehen, was für Funktionen die Register ausführen. Dabei ist die Registernummer, die in der folgenden Aufstellung verwendet wird, gleich mit der Nummer, die im Adress-Register eingetragen werden muß, um das gewünschte Register anzusprechen.

Interessant ist noch die Tatsache, daß das Adressregister seinen Inhalt bis zur nächsten Programmierung behält. Man kann also ohne Probleme mehrmals nacheinander auf ein Datenregister zugreifen, ohne jedes Mal das Adressregister neu laden zu müssen.

Doch jetzt zur Registerbeschreibung.

Reg 0,1 :

Diese Register bestimmen die Periodendauer und damit die Frequenz des Tonsignals an ANALOG A. Allerdings sind nicht alle 16 Bit benutzt. Verwendung finden alle 8 Bit des Register 0 und die vier unteren Bit des Registers 1. Dabei kann die Frequenz mit dem Register 0 fein, mit dem Reg. 1 in groben Stufen beeinflußt werden. Je kleiner der 12-Bit-Wert dieser Register wird, desto höher wird der Ton.

Reg 2,3 :

Funktion wie Reg 0,1, aber Kanal B.

Reg 4,5 :

Funktion wie Reg 0,1, aber Kanal C.

Reg 6 :

Dieses Register beeinflußt mit seinen unteren 5 Bit den Rausch-Generator. Auch hier gilt: je kleiner der Wert im Register, desto höher die Rausch-Frequenz.

Reg 7 :

In diesem Multi-Funktionsregister kontrollieren die einzelnen Bits unterschiedliche Aufgaben. In der folgenden Tabelle werden die Beeinflussungen genannt:

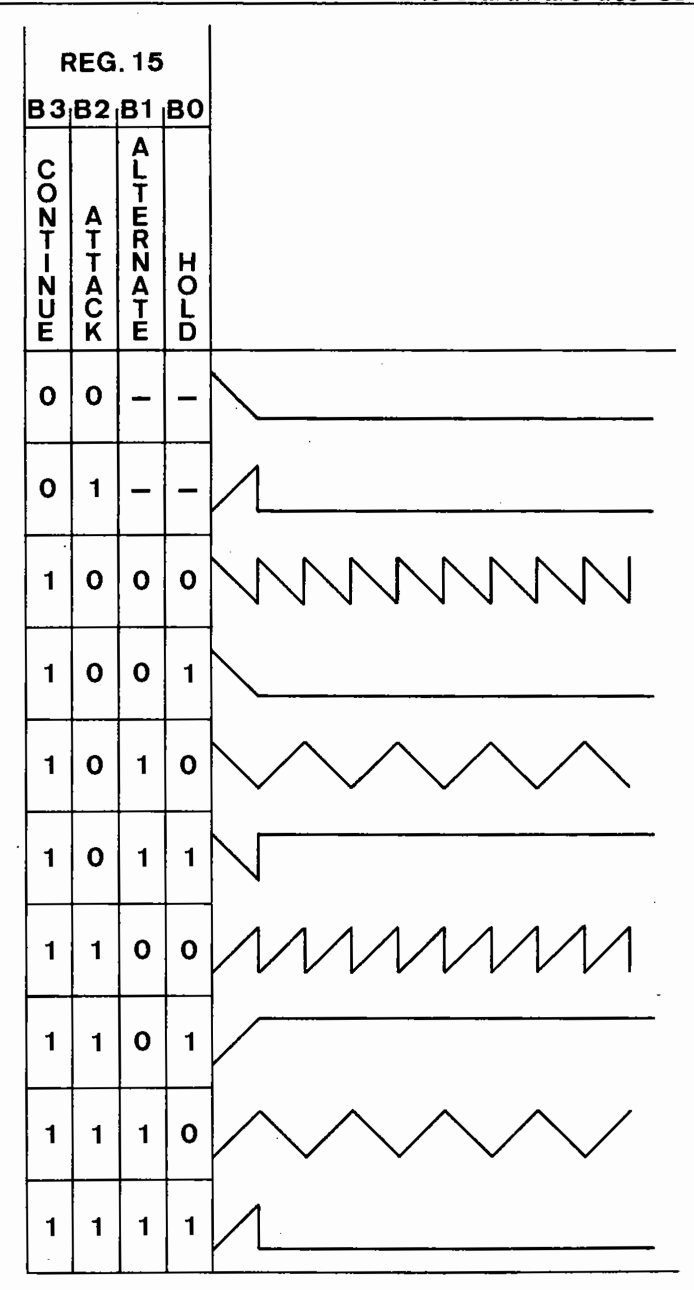
Bit 0 : Ton von Kanal A ein-/ausschalten 0=ein / 1=aus
Bit 1 : Ton von Kanal B ein-/ausschalten 0=ein / 1=aus
Bit 2 : Ton von Kanal C ein-/ausschalten 0=ein / 1=aus
Bit 3 : Rauschen zu Kanal A zu-/abschalten 0=zu / 1=ab
Bit 4 : Rauschen zu Kanal B zu-/abschalten 0=zu / 1=ab
Bit 5 : Rauschen zu Kanal C zu-/abschalten 0=zu / 1=ab
Bit 6 : Port A als Ein-/Ausgang 0=in / 1=out
Bit 7 : Port B als Ein-/Ausgang 0=in / 1=out

Reg 8 :

Dieses Register bestimmt die Lautstärke des Signals an Kanal A. Zur Lautstärkeeinstellung werden die vier unteren Bits verwendet.
Das Bit 4 hat eine besondere Bedeutung. Ist es gesetzt, dann wird die Lautstärke durch das Hüllkurven-Register bestimmt, der Inhalt der Bits 0 bis 3 wird dann ignoriert.

Reg 9 :

Wie Reg 8 für Kanal B.



1.8.2.1 Hüllkurven des PSG

Reg 10 :

Wie Reg 8 für Kanal C.

Reg11,12 :

Alle 16 Bit dieser beiden Register beeinflussen die Periodendauer der Hüllkurve. Der Inhalt des Reg. 11 wird als Low-Byte betrachtet, d.h. er beeinflußt die Periodendauer in feinen Schritten, das Reg. 12 ist das High-Byte des Hüllkurven-Generators.

Reg 13 :

Die Bits 0 bis 3 dieses Registers bestimmen die Kurvenform des Hüllkurven-Generators. Die Zuordnung in Worten verständlich wiedergeben zu wollen, ist fast unmöglich. Die erzeugbaren Hüllkurven sind darum in der Grafik 1.8.2.1 gezeigt.

1.8.3 Der Betrieb des AY-3-8912 im CPC

In diesem Abschnitt wollen wir uns mit dem konkreten Anschluß und einigen mehr praktischen Dingen zum Betrieb des Sound Chip im CPC beschäftigen. Da die vorige Registerbeschreibung das Thema notgedrungen abstrakt und vielleicht nicht sehr anschaulich erläuterte, werden Sie nach Abschluß dieses Kapitels einige Spezialitäten des PSG besser verstehen.

Die Pins 3, 17 und 19 des Soundchip sind auf + 5 Volt gelegt. Über den Pin 3 bekommt der AY-3-8912 seine Betriebsspannung. Da BC2 (Pin 19) und A8 (Pin 17) auf + 5 Volt liegen, gehen sie in die Registerauswahl nicht ein.

Die verbleibenden Register-Steueranschlüsse BC1 (Pin 20) und BDIR (Pin 18) sind mit den Portbits PC6 und PC7 des 8255 verbunden. Je nach Zustand dieser Anschlüsse können dem PSG Registeradressen mitgeteilt sowie Daten in den PSG geschrieben oder herausgelesen werden.

Der eigentliche Adress- und Datentransfer geschieht über die PSG-Anschlüsse D0 bis D7, die mit dem Port A des 8255 verbunden sind. Je nach geforderter Aktion muß der Port A des 8255 als Ein- oder Ausgang programmiert werden.

Das Clock-Signal am Pin 15 ist ein Rechtecksignal mit einer Frequenz von 1 MHz. Dieses Signal wird durch Teilung der Quarzfrequenz vom Gate Array geliefert. Von diesem Signal werden durch Frequenzteilung alle Ton- und Hüllkurvenfrequenzen abgeleitet.

Der I/O-Port des PSG ist verbunden mit der Tastatur und dem Anschluß für den Joystick. Eine detaillierte Beschreibung der Tastatur und des Joysticks finden Sie in einem späteren Kapitel, hier sollen uns nur die klanglichen Möglichkeiten des Sound Chip interessieren.

Die wichtigsten Anschlüsse an diesem IC sind sicherlich die drei Analog-Ausgänge A, B und C an den Pins 1, 4 und 5. Diese Ausgänge sind als sogenannte Open-Emitter-Ausgänge ausgeführt. Um eine Tonwechselspannung ausgeben zu können, werden die drei 1K-Widerstände benötigt, die zwischen Ausgang und Masse geschaltet sind.
Von diesen Widerständen wird das Soundsignal einmal über drei Widerstände gemischt und steht als Mono-Signal am Anschluß 1 des Expansion Connectors zur Verfügung. Dieses Mono-Signal wird aber auch zum internen Verstärker und weiter zum Lautsprecher geführt.
Zusätzlich werden die drei Ausgänge aber noch auf die Stereo-Klinkenbuchse an der Geräterückseite geführt. Dazu wird das Signal des Kanals B über zwei Widerstände gleichermaßen auf die beiden Stereo-Kanäle gelegt. Die Ausgänge A und C werden jeweils direkt über je einen Entkopplungskondensator auf einen der Stereo-Kanäle gelegt. Durch diese Art der Beschaltung sind bei geschickter Programmierung sogar echte Stereoeffekte möglich. Denkbar wäre z.B. einen Ton zunächst nur über den Kanal A auszugeben. Nach einiger Zeit könnte derselbe Ton zusätzlich über den Kanal B ausgegeben werden. Dabei könnte die Lautstärke des Signals an Kanal B langsam steigen, die

Lautstärke des Signals dagegen entsprechend reduziert werden. Durch diese Maßnahmen erscheint es, als würde der Ton von einer Ecke des Raums in die Mitte zwischen die beiden Lautsprecher-Boxen wandern. Von hier kann es jetzt bei Bedarf weiter in die andere Ecke gehen.
Diese Möglichkeiten sind sogar in BASIC mit dem leistungsstarken SOUND-Kommando möglich. Das Bedienungs-Handbuch ist aber bei der Angabe der Verteilung der drei Tonkanäle auf die zwei Stereokanäle widersprüchlich. Beachten Sie dies, wenn Sie Ihren CPC mit einer Stereoanlage verbinden. Nur Klänge des Kanals B erscheinen auf beiden Kanälen der Stereoanlage.

Wie aber erzeugt der PSG eigentlich die Töne? Betrachten wir einmal die Vorgänge an einem Kanal im Detail.
Wie schon erwähnt werden alle Töne vom Clock-Signal an Pin 15 abgeleitet. Zunächst wird das Taktsignal durch 16 geteilt. Daraus resultiert beim CPC eine Steuerfrequenz von 62,5 KHz. Diese Frequenz wird jetzt auf einen programmierbaren Frequenzteiler geführt. Je nach Inhalt der Tongenerator-Register wird die Steuerfrequenz weiter geteilt, um die gewünschte Tonfrequenz zu erhalten.
Dabei haben die Entwickler des ICs besonders tief in die Trickkiste gegriffen. Die Teilerkette besteht nicht nur aus Flip-Flops, die die Frequenz durch zwei teilen können. Durch eine spezielle Schaltungstechnik sind auch ungerade Teilerfaktoren möglich. Die Steuerfrequenz kann durchaus auch durch drei oder 17 geteilt werden. Dadurch erst können gerade im hohen Frequenzbereich alle benötigten Werte erzeugt werden.

Der Inhalt der Tongenerator-Register bestimmt also den Teilerfaktor für das Tonsignal. Wird das Register 0 des PSG mit dem Wert 100, das Register 1 mit dem Wert 0 geladen, so wird die Steuerfrequenz durch 100 geteilt. Am Ausgang der Teilerkette des Kanals A liegt ein Signal mit der Frequenz von 625 Hertz an.
Dieses Signal ist aber noch nicht am Ausgang A abzunehmen. Dazu muß der entsprechende Kanal zunächst eingeschaltet

werden. Dies wird durch Löschen des entsprechenden Bits im Register 7 erreicht. Da wir in unserem Beispiel den Kanal A gewählt haben, müssen wir das Bit 0 löschen. Dabei ist der Zustand der übrigen Bits zu beachten. Beim CPC bedeutet dies konkret, das Bit 6 nicht ungewollt zu verändern, da sonst die Tastatur gesperrt wird.
Aber auch jetzt ist wahrscheinlich noch kein Ton zu hören, da noch die Lautstärke des jeweiligen Kanals einzustellen ist. Für Kanal A ist das Register 8 zuständig. Ein Wert von 1 erzeugt nun einen leisen Ton, bei einem Wert von 15 erhalten wir die maximale Lautstärke.

Setzen wir das Bit 4 im Lautstärke-Register, dann wird die Information in den Bits 0 bis 3 ignoriert. Jetzt bestimmen die Register 11, 12 und 13 die Lautstärke. Allerdings ist die Lautstärke jetzt nicht mehr auf einen Wert fixiert, sondern variabel.

Betrachten wir zunächst das Register 13. Dieses Register trägtdenoffiziellenNamen'**ENVELOPESHAPE/CYCLECONTROL REGISTER**'. Die Funktion wird am besten in einem kleinen Beispiel deutlich.3
Nachdem wir die Register 0, 1, 7 und 8 mit den entsprechenden Werten versorgt haben, schreiben wir einmal in das Reg. 13 den Wert 12. Jetzt sind die Bits 2 und 3 gesetzt, die unteren 2 Bits gelöscht.
Die in der Registerbeschreibung gezeigte Tabelle zeigt bei dieser Kombination eine Folge langsam ansteigender und schnell abfallender 'Zacken'. In der Praxis bedeutet dies, daß die Lautstärke des Tons zunächst langsam bis zum Maximum ansteigt. Dann wird der Ton abgeschaltet und die Lautstärke nimmt wieder zu. Dieser Zustand bleibt erhalten, bis ein neues Kommando zum Register 13 geschickt wird.

Die Zeit des Ansteigens der Lautstärke kann über die Register 11 und 12 eingestellt werden. Diese Register beeinflussen ähnlich den Tongenerator-Registern eine weitere programmierbare Teilerkette im PSG. Die Teilerkette wird mit einem Signal versorgt, das dem durch 256 geteilten

Clock-Signal entspricht. Das ergibt eine Frequenz von 3906.25 Hertz entsprechend einer Periodendauer von etwa 250 Mikrosekunden.

Wird in das Reg. 11 ein Wert 1, in das als High-Byte arbeitende Reg 12 der Wert 0 geschrieben, so wird tatsächlich die Lautstärke des Tones in 250 Mikrosekunden von 0 bis zur maximalen Lautstärke hochgeregelt. Das liegt aber schon im Bereich der hörbaren Töne und erzeugt einen deutlichen Pfeifton, der dem eigentlich gewünschten Ton überlagert wird.
Aus diesem Grund werden die Registerwerte immer deutlich größer gewählt werden. Beim Maximalwert (255 ins Reg 11 und Reg 12) dauert das Ansteigen bis zur vollen Lautstärke ganze 16,8 Sekunden.

Diese Beeinflussung der Lautstärke über die Envelope-Register wird von der Software des CPC nicht verwendet. Das ENV-Kommando beeinflußt die Lautstärke des Tons nur über Manipulationen der unteren vier Bit des Lautstärke-Registers. Das ENT-Kommando des CPC hat im PSG überhaupt kein Equivalent. Diese Funktion wird durch geschicktes Verändern der Tongenerator-Register erzeugt.

1.9 Die Floppy im CPC 664 und 6128

Anders als beim Vorgänger 464 sind beim 664 und 6128 das Floppyinterface und ein Laufwerk in das Gehäuse integriert worden. Daduch wird nicht nur der Expansion-Slot auf der Rückseite für weitere Peripherie besser zugänglich. Auch der auf dem Arbeitsplatz benötigte Raum und die Anzahl der 'Strippen' wird reduziert.
Grundsätzlich sind die Schaltungen in allen drei Rechnern sowohl funktions- wie auch software-mäßig vollständig kompatibel. Die gesamte Literatur zu den SCHNEIDER-Floppy-Laufwerken (z.B. DATA BECKER, Das Große Floppybuch) ist also für alle Rechner zutreffend, so daß auch die Besitzer eines CPC 664 oder 6128 von den bisher erschienenen Büchern profitieren können.

Den Mittelpunkt des Controller-Boards bildet der integrierte Floppy Disk Controller (FDC) uPD 765. Dieses IC stellt die Schnittstelle zwischen den Laufwerken und dem Prozessor des CPC dar. Zwar kann man Floppystationen auch ohne FDC aufbauen, durch die hohe 'Eigenintelligenz' des FDC vereinfacht sich die Konstruktion jedoch wesentlich. Der nötige Hardwareaufwand, aber auch das Ausmaß der zu schreibenden Betriebssoftware wird durch den Einsatz eines FDC drastisch reduziert. Ein Beispiel mag dies verdeutlichen.
Die Floppystation 1541 der Firma Commodore, vielen von Ihnen sicher bekannt als Floppystation zum Commodore C64, ist eine ohne FDC aufgebaute Floppystation. Abgesehen von der konstruktiv bedingten langsamen Geschwindigkeit der Datenübertragung (über die Sie als CPC-Besitzer nur lächeln können) ist der Aufwand der Hardware für dies Laufwerk deutlich höher als bei der CPC-Floppy. Die Digitalelektronik der 1541 enthält einen eigenen Prozessor, zwei 40-polige Peripherie-IC und jede Menge verschiedener TTL-ICs. Ein kompletter CPC 664 enthält eine vergleichbare Menge an Bauteilen!
Die Betriebssoftware für die 1541 ist mit etwa 16 K doppelt so groß wie das AMSDOS. Keine Frage, daß Entwickler (aus Gründen der Bequemlichkeit) und Kaufleute (aus Kosten-

gründen) gern zu den komfortabel einzusetzenden FDCs greifen.

Insgesamt besteht die Hardware des Controllers aus nur einer Handvoll Bauteilen. Die erstaunlich geringe Anzahl von Bauteilen konnte nur durch die hohe Integration von drei ICs erreicht werden. Gemeint sind der FDC, der Datenseparator und das ROM mit dem AMSDOS.

Das 16 KByte große AMSDOS-ROM enthält in ca. 8 KByte alle wesentlichen Routinen, die um Betrieb der Floppy nötig sind. Dieses ROM in seinem 28-poligen Gehäuse enthält außerdem in den verbleibenden 8 KByte einen Teil des auf Diskette gelieferten CP/M 2.2-LOGO Interpreters.

1.9.1 Der FDC 765

Der von der Firmen NEC als uPD 765, von ROCKWELL als R 6765 und von INTEL als 8765 vertriebene FDC kann als ein hoch spezialisierter Mikroprozessor angesehen werden. Die Möglichkeiten dieses ICs sind so umfangreich und komplex, daß diese Bezeichnung sicher nicht zu hoch gegriffen ist.

Das vom FDC benutzte Datenformat entspricht dem IBM-Format 3740 in Single Density und IBM System 84 in Double Density. Durch diese Festlegung können z.B. Commodore- oder Apple-Disketten leider nicht gelesen oder beschrieben werden.
Mit seinen 40 Pins stellt er alle für den Betrieb handelsüblicher Laufwerke der Größen 8", 5 1/4" und 3" benötigten Signale zur Verfügung. Durch die vorhandenen Steuersignale ist der Entwickler in der Lage, diesen FDC an fast jeden Prozessor anzuschließen. Dabei sind zwei grundsätzliche Möglichkeiten zum Anschluß und Betrieb gegeben. Die erste Methode ist der DMA-Betrieb. Zusammen mit einem DMA-Controller kann der FDC für den Datentransfer beim Lesen und Schreiben die Kontrolle über den Speicher des Computersystems übernehmen. Er holt sich dann mit Hilfe des DMA-Controllers benötigte neue Daten aus dem Speicher oder

schreibt, ebenfalls unter Umgehung des Prozessors, die von der Diskette gelesenen Werte in den Speicher. Diese sehr schnelle Methode des Datentransfers wird aber im CPC nicht eingesetzt und wird hier nur der Vollständigkeit halber erwähnt.

Bei der zweiten, im CPC eingesetzten Methode wird der Datentransfer vom Prozessor übernommen. Bei dieser zweiten Methode muß aber wiederum zwischen zwei Möglichkeiten des Betriebs des FDC unterschieden werden.
Da wäre zunächst die Interrupt-Methode. Hierbei wird zu jedem Datentransfer ein Interrupt erzeugt. In der entsprechenden Interupt-Routine des Prozessors muß dann das nächste Daten- oder Befehlsbyte vom Prozessor geliefert oder gelesen werden. Durch den Hardwareaufbau des CPC kam diese Methode wohl auch nicht in Betracht, so daß die Entwickler zur Polling-Methode gegriffen haben. Hierbei muß der Prozessor regelmäßig in Registern des FDC prüfen, welche Aktion als nächstes vom FDC gefordert wird.

Doch betrachten wir zunächst einmal die Leistungsdaten des 765 im Überblick. Bedenken Sie jedoch, daß die Entwickler des Controller Boards nicht alle Möglichkeiten des 765 genutzt haben.

programmierbare Sektorlänge
alle Laufwerkdaten programmierbar
bis zu vier Laufwerke anschließbar
Datentransfer wahlweise im DMA- oder Nicht-DMA-Modus
anschließbar an fast alle gängigen Prozessortypen
einfache 5-Volt-Stromversorgung
einfacher Ein-Phasen-Takt von 4 oder 8 MHz
40-poliges IC-Gehäuse

Dem letzten Punkt dieser kurzen Aufstellung wollen wir uns jetzt etwas detaillierter zuwenden.

1.9.2 Die Anschluß-Belegung des FDC

Die Anschlüsse des FDC 765 lassen sich in verschiedene Gruppen unterteilen. Die erste Gruppe von Anschlüssen stellen das Interface zum Systemprozessor dar. Über diese Anschlüsse wird also die Steuerung des FDC vom Prozessor aus vorgenommen.

Die zweite Gruppe ist nur in Verbindung mit dem DMA-Betrieb nötig. Über diese Signale kommunizieren DMA-Controller und FDC.

Das Interface zu den Floppy-Laufwerken wird von der dritten, mit 19 Anschlüssen zahlenmäßig stärksten Gruppe der Anschlüsse geliefert.

In der vierten und letzten Gruppe lassen sich die Anschlüsse für die Stromversorgung und den Takt zusammenfassen.

Beginnen wir die Betrachtung der Anschlüsse mit der ersten Gruppe, dem Interface zum Prozessor.

Das Prozessor-Interface

RESET :

Der RESET-Anschluß des FDC ist High-aktiv. Im normalen Betrieb liegt dieser Anschluß auf Masse-Potential. Durch ein High am RESET-Pin wird der FDC in einen definierten Zustand gebracht.

CS* : CHIP SELECT.

Durch ein Low an diesem Pin wird der FDC selektiert. Erst bei CS* = Low werden RD* und WR* für den FDC gültig. Da die Erzeugung des CS dem Entwickler freigestellt ist, kann der FDC wahlweise Memory-Mapped, also als Bestandteil des Speicherbereichs, oder über Portadressen angesprochen werden.

RESET	1	VCC (+5V)
AD		RW/SEEK
WR		LCT/DIR
CS		FLTR/STEP
A 0		HOLD
DB 0		READY
DB 1		WPRT/2 SIDE
DB 2		FLT/TRKO
DB 3		PS 0
DB 4		PS 1
DB 5		WDATA
DB 6		US 0
DB 7		US 1
DRQ		SIDE
DACK		MFM
TC		WE
INDEH		SYNC
INT		RDATA
Ø		WINDOW
GND		WCLK

1.9.1.1 Pinout des FDC 765

RD* : READ*

Dieser Anschluß muß mit dem RD*-Signal des Prozessors verbunden werden. Wann immer der Prozessor Daten aus dem FDC lesen will, wird diese Leitung auf Low gelegt.

WR* : WRITE*

So, wie die RD*-Leitung Lesezugriffe des Prozessors signalisiert, so zeigt ein Low an WR*, daß der Prozessor Daten oder Befehle in den FDC schreibt.

A0 : ADRESS LINE 0

Der FDC verfügt über nur zwei von außen ansprechbare Adressen. Die Unterscheidung zwischen den beiden Adressen wird mit dem Signal A0 vorgenommen. Diese Leitung ist normalerweise mit dem niedersten Adressbit des Prozessors verbunden.

DB0 - DB7 : DATABUS 0-7

Diese Anschlüsse des FDC werden mit dem Systemdatenbus verbunden. Alle Kommandos und Daten werden über diese acht bidirektionalen Anschlüsse transportiert. Die jeweilige Datenrichtung wird dabei entweder vom Prozessor oder im DMA-Mode vom DMA-Controller bestimmt.

INT : INTERRUPT

Über diesem Anschluß kann der FDC einen Interrupt des Systemprozessors erzeugen. Interrupts werden zu jedem Byte-Transfer erzeugt (im CPC nicht angeschlossen).

Signale für den DMA-Modus (im CPC nicht verwendet)

DRQ : DMA REQUEST

Über diesen Anschluß signalisiert der FDC dem DMA-Controller, daß ein Speicherzugriff erfolgen soll. Bei der nächsten möglichen Gelegenheit übernimmt daraufhin der DMA-Controller den Systembus. Der Prozessor wird dabei abgeschaltet.

DACK* : DMA ACKNOWLEDGE

Mit diesem Signal wird dem FDC angezeigt, daß der DMA-Controller den Bus übernommen hat und jetzt mit dem Datentransfer begonnen hat.

TC : TERMINAL COUNT

Durch einen High-Pegel an diesem Anschluß wird der Datentransfer von und zum FDC unterbrochen. Obwohl dieser Anschluß in der Hauptsache im DMA-Modus verwendet wird, kann auch in interruptgesteuerten Systemen der Datentransfer über diesen Anschluß unterbrochen werden.

Das Floppy-Interface

US0, US1 : UNIT SELECT 0/1

Über diese beiden Anschlüsse können direkt zwei, mit Hilfe eines Zwei-zu-Vier-Decoders jedoch vier Laufwerke angeschlossen werden. Über diese Anschlüsse wird das jeweils gewünschte Laufwerk zum Lesen oder Schreiben von Daten angesprochen.

HD : HEAD SELECT

Da der FDC für den Betrieb von Doppelkopf-Laufwerken vorbereitet ist, kann bei Verwendung dieser Drives die Kopfauswahl über diesen Anschluß geschehen.

HDL : HEAD LOAD

Dieses Signal wird fast ausschließlich bei 8"-Laufwerken eingesetzt. Die Motoren dieser Laufwerke werden nicht bei Bedarf eingeschaltet, sondern laufen normalerweise immer. Um nun Diskette und Schreiblesekopf zu schonen, wird der Kopf nur bei Bedarf über einen Hubmagneten 'geladen', also an die Diskettenoberfläche gebracht. Die Steuerung des Hubmagneten wird dann mittels HDL vorgenommen.

IDX : INDEX

An diesem Anschluß wird das von der Index-Lichtschranke erzeugte Signal angelegt und signalisiert dem FDC den physikalischen Anfang eines Tracks.

RDY : READY

Das von der Floppy gelieferte Signal READY zeigt an, daß sich im Laufwerk eine Diskette befindet und daß sich diese mit einer gewissen Mindestgeschwindigkeit dreht. Erst nach Erscheinen des READY greift der FDC auf das Laufwerk zu.

WE : WRITE ENABLE

Dieser Ausgang des FDC muß High sein, um Daten auf die Diskette schreiben zu können.

RW/SEEK : READ WRITE/SEEK

Insgesamt liefert ein Floppylaufwerk mehr Signale, als bei einem 40-poligen Gehäuse für das Floppy-Interface zur Verfügung stehen. Allerdings werden nicht zu allen Zeiten alle Signale gleichzeitig benötigt. Acht dieser Laufwerkssignale hat man darum in zwei Gruppen aufgeteilt, die wahlweise an vier Anschlüsse des FDC gelegt werden können. Über den Anschluß RW/SEEK wählt der FDC selbsttätig die jeweils benötigten Signale aus.

FR/STP : FIT RESET/STEP

Dies ist das erste der vier Doppelsignale am FDC. Dieser Ausgang hat je nach ausgeführter Operation verschiedene Bedeutungen. Einmal kann mit diesem Anschluß das bei einigen Laufwerken vorhandene Fehler-Flip-Flip zurückgesetzt werden. Die zweite, weitaus häufigere Verwendung ist die Ansteuerung des Step-Eingangs des Laufwerks. Zu jedem Kopfwechsel werden die benötigten Impulse an diesem Anschluß geliefert.

FLT/TR0 : FAULT/TRACK0

Auch dieser Eingang kann zwei verschiedene Signale auswerten. Wird eine SEEK-Operation (siehe Programmierung des FDC) durchgeführt, dann wird an diesem Anschluß das Track0-Signal des Laufwerks erwartet. Dies Signal wird durch eine Lichtschranke oder einen mechanischen Schalter erzeugt, wenn der Schreib/Lesekopf auf der physikalischen Spur 0 steht. Die zweite Funktion, das Fault-Signal, wird von einigen Laufwerken im Fehlerfall generiert und kann vom FDC mit dem zuvor beschriebenen Signal FR/STP wieder gelöscht werden. Dieses Signal wird bei Read/Write-Operationen des FDC überprüft.

LCT/DIR : LOW CURRENT/DIRECTION

Die Step-Impulse von FR/STP geben ja nur an, daß der Kopf bewegt werden soll. LCT/DIR bestimmt dazu im Seek-Modus die Richtung der Kopfbewegung. Die Funktion LOW CURRENT wird beim Schreiben der Daten benötigt. Durch dieses Signal läßt sich der Schreibstrom auf den inneren Spuren verringern. Einzelheiten zu diesem Signal finden Sie in der Beschreibung der theoretischen Grundlagen der Diskettenspeicherung.

WP/TS : WRITE PROTECT/TWO SIDE

Unabhängig von den unterschiedlichen Methoden bei den diversen Laufwerksgrößen wird der Zustand des Schreibschutzes als Signal vom Laufwerk an den Controller gemeldet. Dies Signal wird vom Eingang WP/TS bei Schreib/Leseoperationen überprüft. Das Signal TS wird bei Seek-Operationen überprüft. Es wird nur in Verbindung mit Doppelkopflaufwerken benötigt.

WDA : WRITE DATA

Über diesen Anschluß werden die seriellen Schreib-Daten an das Laufwerk geliefert. Das können sowohl die beim Schreiben eines Sektors

vorkommenden Daten wie auch alle beim Formatieren benötigten Informationen sein.

PS0, 1 : PRE SHIFT0/1

Über diese Anschlüsse teilt der FDC bei Double Density-Format (MFM) einer geeigneten Elektronik mit, wie der serielle Datenstrom auf die Diskette geschrieben werden muß. Möglich sind die drei Zustände EARLY, NORMAL und LATE für die Precompensation.

RD : READ DATA

Über diesen Eingang werden die von der Diskette gelesenen Informationen in den FDC gegeben. Aus diesem seriellen Bitstrom werden die ursprünglich geschriebenen Bytes zurückgewonnen.

RDW : READ DATA WINDOW

Dieses Signal wird aus den gelesenen Daten in einem Datenseparator gewonnen.

VCO : VCO SYNC

Dieses Signal wird zur Steuerung des VCO im PLL-Datenseparator benötigt.

MFM : MFM MODE

Dieser Anschluß signalisiert, ob der Controller im Single Density-Format (MF) oder im Double Density-Format (MFM) arbeitet.

Stromversorgung und Taktsignale

Vcc : +5 Volt

Über diesen Anschluß erhält der FDC seine Versorgungsspannung. Die Spannung von 5 Volt sollte im Bereich von +- 5% konstant sein. Der benötigte Strom des FDC beträgt max. 150 mA.

GND : GROUND

Masseanschluß des FDC

CLK : CLOCK

Der FDC benötigt einen Takt. Je nach Laufwerken muß dieser Takt 4 MHZ (bei 5 1/4" und kleiner) oder 8 MHz (bei 8") betragen.

WCK : WRITE CLOCK

Die Frequenz dieses Signals muß je nach gewähltem Datenformat gewählt werden. Bei MF muß der Takt 500 kHz, bei MFM 1 MHz betragen. Diese Frequenz bestimmt die Übertragungsgeschwindigkeit der Daten von und zur Floppy.

1.9.3 Einsatz des FDC 765 im CPC

Leider haben die Entwickler lange nicht alle Möglichkeiten des FDC genutzt. So können nur zwei statt der möglichen vier Laufwerke angeschlossen werden. Auch der Betrieb von Doppelkopf-Laufwerken ist nicht möglich, da das HEAD-SELECT-Signal zwar herausgeführt, aber nicht benutzt wird. Schlimmer noch ist es dem Signal HEAD LOAD ergangen, es ist nirgends angeschlossen. Dieser Mangel lässt sich jedoch verschmerzen, da ein Betrieb von 8"-Laufwerken nicht nur für den 'durchschnittlichen' Anwender durch die enormen physikalischen Ausmaße dieser Drives uninteressant, sondern auch durch weitere Schaltungsdetails im Controller unmöglich ist.
Trotz dieser Einschränkungen ist der Controller für den Verwendungszweck, dem problemlosen Betrieb zweier 3"-Laufwerke, sehr gut durchdacht konstruiert. Mit nur minimalem Hardware-Aufwand ist ein Controller geschaffen worden, der durchaus exzellente Leistungsdaten bietet.

Bei aller Sparsamkeit der Entwickler hat man zudem dankenswerter Weise nicht die Zuverlässigkeit des Gerätes eingeschränkt. Als Datenseparator wurde in den CPC's das für

diesen Zweck excellent geeignete IC SMC 9216 eingesetzt. Dieser Punkt ist insofern wichtig, als der Datenseparator ganz wesentlich für die Rückgewinnung der Daten aus dem Impulsstrom der Floppy und damit für fehlerfreies Lesen der Disketten verantwortlich ist.

Obwohl der DMA-Betrieb die einfachste und eleganteste Methode darstellt, den Floppy-Controller anzuschließen, hat man sich, wohl aus Kostengründen, für einen anderen Weg entschieden. Der FDC wird gepolled. Das bedeutet, daß der Prozessor an Hand des Haupt-Status-Registers den Datentransfer synchronisiert. Die vom Controller erzeugten Interrupts werden nicht benutzt. Tatsächlich ist der Interrupt-Anschluß des FDC nicht beschaltet.

Adressmäßig liegt der FDC auf den Portadressen **&FB7E** und **&FB7F**. Auf der ersten Adresse befindet sich das Haupt-Status-Register, die zweite Adresse gehört zum Datenregister.
Eine dritte Adresse wird vom Controller belegt. Auf dem Port **&FA7E** befindet sich ein Flip-Flop, über das die Laufwerksmotoren gesteuert werden. Schreibt man auf diesen Port eine 1 (OUT &FA7E,1 in BASIC), so werden die Motoren aller angeschlossenen Laufwerke eingeschaltet, schreibt man dagegen eine 0, so werden die Motoren wieder ausgeschaltet.

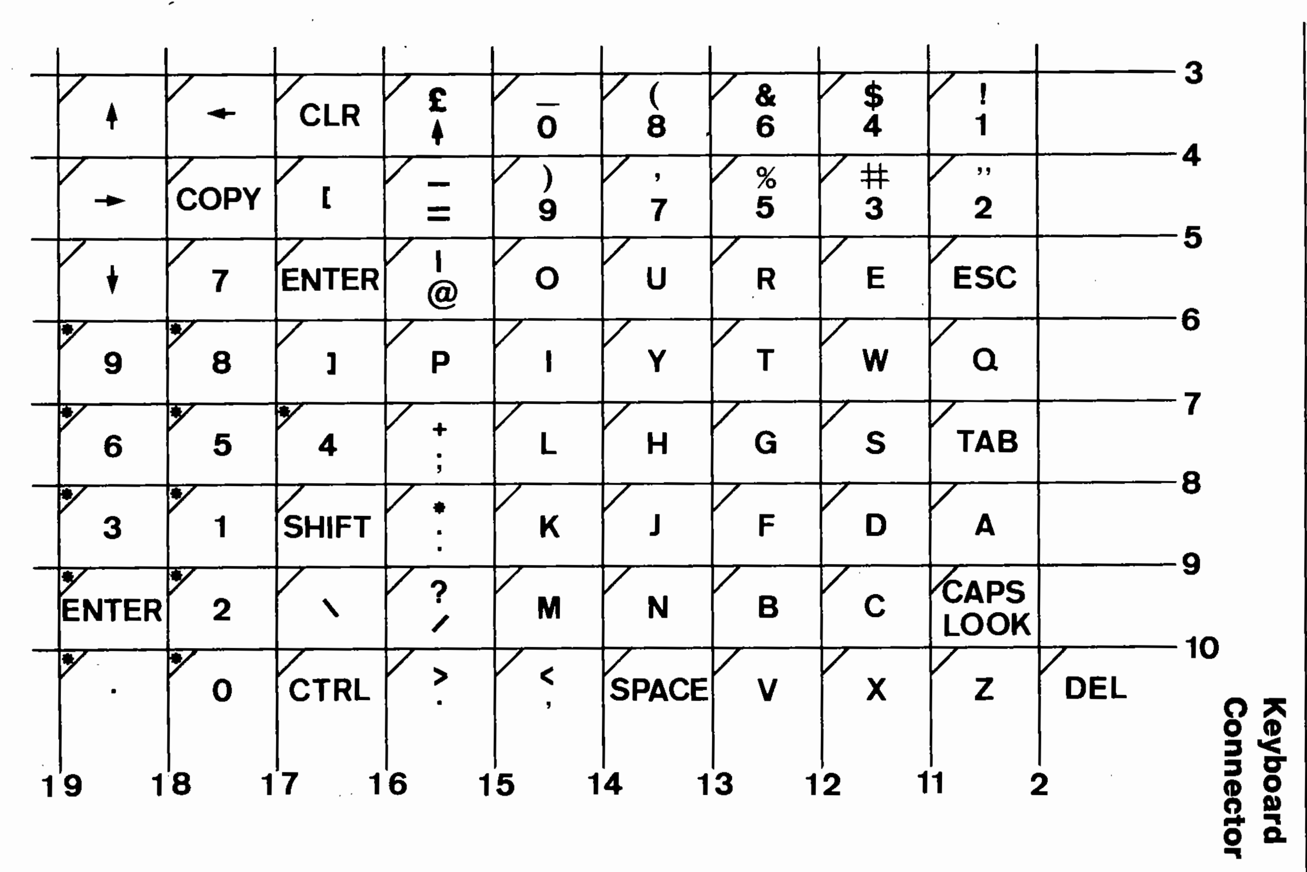
1.10 Die Schnittstellen des CPC

Der Begriff Schnittstelle läßt sich definieren als Verbindungsstelle zwischen Computer und Außenwelt. Dabei kann die Außenwelt sowohl ein anderer Computer, ein Drucker oder sonstige Peripherie, ein Meßgerät oder auch der Mensch sein. Nach dieser Definition von Außenwelt wollen wir in diesem Kapitel nicht nur die an der Geräte-Rückseite angebrachten Steckverbindungen beschreiben, sondern auch Tastatur, Monitor- und Recorderanschluß mit einbeziehen.
Die für den Benutzer wichtigsten Schnittstellen sind Tastatur und Monitor, da diese den unmittelbaren Kontakt zum Computer darstellen. Fangen wir darum mit diesen beiden an.

1.10.1 Die Tastatur

Insgesamt sind auf der CPC-Tastatur 74 Tasten untergebracht. Da die beiden SHIFT-Tasten parallel geschaltet sind, sind also 73 einzelne Tasten abzufragen.
Die Matrix, in der die Tasten angeordnet sind, besteht aus 8 mal 10 Leitungen. Da auch die Joysticks über diese Matrix abgefragt werden, werden insgesamt 79 Tastenpositionen belegt. Der zweite Joystick, über die Buchse im ersten angeschlossen wird, aber nicht auf eigene Positionen der Matrix geführt, die zugehörigen Schalter sind zu Tasten der Tastatur parallel geschaltet. Einen überblick über die Matrix gibt Ihnen die Zeichnung 1.10.1.1
Hardwaremäßig wird die Tastatur über den 8255 und den Sound Chip abgefragt. Das funktioniert im einzelnen etwa folgendermaßen. Der 8255 liefert an den Portausgängen PC0 bis PC3 ein Halbbyte (Nibble), das durch einen Decoder 74LS145 in eine dezimale Information gewandelt wird. Je nach anliegender Eingangsinformation wird einer der zehn Ausgänge Low. Dieser Decoder wird darum auch BCD-Dezimal-Decoder genannt. Liegt die Eingangsinformation nicht im Bereich von 0 bis 9, dann liegen alle Ausgänge auf High.

Der Parallel-Port des Sound Chip ist für die Tastatur-Abfrage als Eingangsport programmiert. Liegt an



1.10.1.1 Die Tastaturmatrix

diesen Eingängen kein Signal an, dann erhält man beim Lesen des Ports an allen Eingängen eine 1, insgesamt also &FF.

Es sei jetzt einmal die Eingangsinformation des Decoders &04. Entsprechend wird der Ausgang Pin 5 Low. Davon nimmt der Port des Sound Chip aber so lange keine Notiz, wie keine entsprechende Taste gedrückt wird. Ein Druck auf die ESC-Taste z.B. hat zu diesem Zeitpunkt keine Auswirkung, da der Ausgang Pin 8 des Decoders High ist. Wird aber die SPACE-Taste gedrückt, dann ändert sich der vom Sound Chip gelieferte Wert. Jetzt liegt durch die gedrückte Taste das Bit 7 des Port an Masse und wir erhalten den Wert &7F vom Sound Chip.

Fünfzigmal in der Sekunde werden alle Tasten einmal überprüft. Dazu werden nacheinander an die vier verwendeten Ausgänge des Port C die Werte 0 bis 9 ausgegeben und nach jeder Ausgabe der Wert des Sound Chip geprüft. Werden dabei irgendwelche gedrückten Tasten registriert, so werden die gedrückten Tasten in einer Tabelle gespeichert und bei Bedarf in Tastennummern und die entsprechenden Zeichen umgerechnet.

Sehr angenehm an der Tastatur ist die Tatsache, daß bis zu 20 Zeichen zwischengespeichert werden. In Basic-Programmen kann man schon Eingaben machen, während der Computer noch Berechnungen vornimmt oder mit der Bildschirmausgabe beschäftigt ist. Nur bei Benutzung des Recorders und beim Listen von BASIC-Programmen sowie bei einigen Diskettenoperationen ist die Tastaturabfrage gesperrt, da die dafür benötigte Zeit nicht zur Verfügung steht. Einzige Ausnahme ist die ESC-Taste, die ja möglicherweise zum Abbruch der Operation benötigt wird.

Übrigens gibt es bei der Tastatur eine kleine Besonderheit. Drücken Sie doch einmal gleichzeitig die Tasten J, K und L. Zur großen Überraschung erscheint auch noch ein H auf dem Bildschirm. Dies passiert immer, wenn drei Tasten gedrückt werden, die die Ecken eines Vierecks in der Tastatur-Matrix

bilden, also auch bei 123 oder DFG. In diesem Fall erscheint gleichzeitig das vierte Zeichen der Matrix.
Dieser 'Fehler' ist nicht weiter schwerwiegend, allerdings können Programme auch durch gleichzeitiges Drücken der Tasten 2,3 und E beendet werden.

1.10.2 Der Video-Anschluß

Der Video-Anschluß des CPC stellt alle Signale für den Betrieb eines Monitors zur Verfügung. Dabei ist es unerheblich, ob es sich um den mitgelieferten Monitor oder einen (fast) beliebigen anderen handelt.

Das Gate Array liefert für den Monitor vier Signale. Drei Signale enthalten die Information über die Farbe, das vierte Signal ist eine Mischung aus den CRTC-Signalen V-Sync und H-Sync.
Diese Signale werden mit Widerständen gemischt und mit einem Transistor verstärkt. Das entstehende Ausgangssignal hat die Bezeichnung LUM und dient den grünen Monitoren als Video-Signal. Aber auch handelsübliche Farbmonitore mit einfachem Video-Eingang können über dieses Signal bei Darstellung aller Farben betrieben werden.

1.10.3 Der Floppyanschluß

Nicht nur die CPC-Rechner, auch die mitgelieferten Handbücher sind besser und umfangreicher als bei vielen Konkurrenzprodukten. Allerdings haben sich hier und da einige kleine Fehler eingeschlichen. So sind z.B. im CPC 6128-Handbuch die an der Geräterückseite vorhandenen Anschlüsse wie beim 664 dargestellt. Dies ist bei den anderen Anschlüssen nicht von Bedeutung, da sowohl die Anschlußzahl als auch die Belegung bei den Rechnern gleich ist. Nur beim Floppyanschluß gibt es kleine Unterschiede. Der Anschluß des 664 ist ein 34-poliger Leiterplattenverbinder. Im 6128 dagegen ist eine 36-polige Centronics-Buchse eingebaut. Von dieser Buchse sind die Anschlüsse 1 und 19 nicht belegt. Da bei den

Centronics-Verbindern die Anschlüsse anders als beim Leiterplattenverbinder gezählt werden, stimmt leider die Bezeichnung der Anschlüsse nicht mit den Angaben im Handbuch überein. Die physikalische Anordnung der Anschlüsse ist aber mit der im Handbuch gezeigten identisch. Einen Adapter für ein beliebiges zweites Laufwerk kann man sich sehr einfach aus einem Stück 34-poligen Flachbandkabel, einem Centronics-Stecker in Quetsch-Ausführung und dem entsprechenden Stecker für das Laufwerk selbst herstellen. Damit lassen sich dann auch 5 1/4"-Laufwerke anschließen.
Die Anschlüsse des Floppy-Verbinders sollen an dieser Stelle nicht mehr separat besprochen werden, da die Beschreibung schon im Kapitel über den Floppy-Controller erfolgt ist.

1.10.4 Der Recorder

Obwohl in Ihrem CPC bereits ein Floppy-Laufwerk eingebaut ist, verfügt der Rechner über einen Anschluß zum Betrieb eines Cassettenrecorders. Dadurch wird zum einen die Möglichkeit zur Nutzung der vorhandenen CPC 464-Software gegeben, zum weiteren jedoch steht mit dem Cassettenrecorder ein sehr preiswertes Backup-Medium zur Verfügung. Auch wird die Kompatibilität zwischen den verschiedenen CPC-Maschinen mit diesem Anschluß gewahrt.
Als Recorder kann jedes handelsübliches Laufwerk angeschlossen werden. Wichtig ist nur, daß ein ausreichender Signalpegel an der Ohrhörer-Buchse vorhanden ist, und daß der Recorder nicht zu sehr 'leiert'. Zu starke Gleichlaufschwankungen stöhren das besonders bei hoher Übertragung einzuhaltene Timing.

Doch wenden wir uns dem verwendeten Aufzeichnungsformat zu. Grundsätzlich kann der Recorder wie auch die Diskette die Daten nur bitweise speichern. Jedes zu speichernde Byte muß also in die einzelnen Bits zerlegt und übertragen werden. Anders als bei der Diskette, wo die Zerlegung durch den Controller übernommen wird, muß die Zerlegung der Cassettendaten vom Prozessor per Software vorgenommen werden, wobei zuerst das höchstwertige Bit zum Recorder geschickt wird.

Das vom 8255 gelieferte Signal für den Recorder ist ein Rechtecksignal. Jedes Bit wird als eine Rechteck-Schwingung aufgezeichnet, bei der die Low-Phase genau so lang ist wie die High-Phase. Man sagt auch, das Rechteck-Signal habe ein Tastverhältnis von 1:1. Ein 0-Bit benötigt die halbe Zeit eines 1-Bits.
Aus diesem Grund sind die Angaben über die Aufzeichnungsgeschwindigkeit auch nur ungefähre Angaben. Es ist offensichtlich, daß ein Datenblock aus lauter 0-Bytes in der Hälfte der Zeit gespeichert und geladen werden kann, wie ein ebenso langer Block, der ausschließlich aus &FF besteht. Da aber statistisch die Verteilung von 0- und 1-Bits in einem Datenblock etwa gleich ist, kann man vòn den Angaben 1000 Baud (1 Baud = 1 Bit pro Sekunde) bei SUPER-SAVE (SPEED WRITE 0) und 2000 Baud bei SPEED-LOAD (SPEED WRITE 1) ausgehen.

Jede Cassetten-Datei, unabhängig, ob es sich um Programme oder Daten handelt, kann maximal 65536 Bytes lang sein. Die Dateien werden in Blocks übertragen, die jeweils 2048 Bytes enthalten. Jeder Block enthält maximal acht 256 Bytes große Datensegmente. Vor jedem Block wird ein Header, also ein Kopf oder Vorspann, übertragen.

Obwohl es keine elektrische Verbindung zum Verstärker und Lautsprecher gibt, kann bei aufgedrehtem Lautstärkeregler das Laden und Abspeichern von Daten und Programmen verfolgt werden.

Der Header der Blocks ist akustisch einfach zu identifizieren. Es ist der zu Beginn eines jeden Blocks hörbare lange gleichmäßige Ton sowie einige folgende Bytes, die aber mit dem Ohr nicht zu unterscheiden sind.
Der lange, gleichmäßige Ton ist eine Serie von 2048 1-Bits. Nach diesen Bits folgt ein einziges 0-Bit und darauf ein Synchronisationsbyte. Die lange Folge der 1-Bits zu Beginn wird vom Rechner benötigt, um die Aufnahme-Baud-Rate zu bestimmen. Das 0-Bit zeigt dem Rechner, daß dieser Vorspann beendet ist und das Sync-Byte wird benötigt, um zwischen der Header-Information und den Daten zu unterscheiden.

Die Header-Information steht in einem 64 Byte langen Datenbereich, der vor jedem 2K-Daten-Block übertragen wird. In diesem Header-File finden sich Informationen über die eigentliche Datei, z.B. der Name, ob das File geschützt ist oder nicht, ob es sich um ein Basic-Programm oder eine Ascii-Datei handelt und wie lang das Programm ist.

Der genaue Aufbau dieses Header ist folgendermaßen:

Bytes 0-15 :
Name der Datei, wenn kürzer als 16 Bytes, dann mit 00 aufgefüllt.

Byte 16 :
Block-Nummer, in diesem Byte steht die Nummer, die beim Laden oder auch beim Catalog angezeigt wird.

Byte 17 :
Steht in diesem Byte ein anderer Wert als 00, dann handelt es sich um den **letzten Block der Datei.**

Byte 18 :
Dieses Byte enthält den **File-Typen**. Die Information ist in den einzelnen Bits verschlüsselt. Die Bedeutung der Bits folgt im Anschluß an diese Tabelle.

Bytes 19,20 :
In diesen Bytes ist die **Länge der File-Information dieses Blocks** enthalten. Ist der Block, also die 2 K, voll beschrieben, so enthalten diese Bytes den Wert &0800, beim letzten Block oder bei Programmen, die kürzer als 2 K sind, ist hier die Anzahl der Bytes des Blocks enthalten.

Bytes 21,22 :
Diese Bytes geben die **Ladeadresse** an, von wo die Daten ursprünglich geschrieben wurden. Bei Basic-

Programmen ist das die Adresse 368 Dezimal, bei Binär-Files, also Maschinensprache, normalerweise die Adresse, an der das Programm im Speicher läuft.

Byte 23 :

Ist der Inhalt dieses Bytes ungleich Null, dann handelt es sich bei dem Block um den **ersten Block des Files.**

Bytes 24,25 :

In diesen Bytes ist die **Länge des Files** enthalten.

Bytes 26,27 :

Wird ein Maschinenprogramm mit 'RUN "filename"' gestartet, so wird der Inhalt dieser Header-Bytes als **Startadresse eines Maschinensprache-Files** interpretiert. Das Programm wird also automatisch an der angegebenen Adresse gestartet.

Die restlichen Bytes 28 bis 63 des Header werden nicht vom Betriebssystem genutzt und stehen dem versierten Programmierer zur Verfügung.

Doch jetzt die Aufschlüsselung der Bits im Byte 18 des Header.

Bit 0 :

Ist dieses Bit gesetzt, so ist das entsprechende File als geschützt erklärt. Geschützte Programme können von Basic aus mit 'SAVE "NAME",p' erzeugt werden.

Bit 1-3 :

Diese Bits bestimmen den Typ des Files. Obwohl mit drei Bit acht verschiedene File-Typen möglich sind, werden nur die File-Typen Basic-Prg (0), Binärfile (1) und Asciidatei (3) verwendet.

Bit 4-7 :
In diesen Bits ist normalerweise eine 0 zu finden, nur Asciidateien haben im Bit 4 eine 1.

Wie bereits erwähnt wird die gespeicherte Information in den einzelnen Blocks weiter unterteilt zu einzelnen Segmenten. Jedes Segment besteht aus 256 Daten-Bytes und Checksummen-Bytes. Die Checksumme jedes Segments wird nach einer speziellen Formel berechnet und erlaubt es, beim Lesen des Files zu prüfen, ob die Bits ordnungsgemäß übertragen wurden. Sobald die errechnete Checksumme nicht mit den gelesenen Werten übereinstimmt, wird der READ ERROR B angezeigt.
Der READ ERROR A zeigt an, daß ein Bit gelesen wurde, dessen Zeit zu lang für die errechneten Werte für Null- oder Eins-Bits ist. Dieser Fehler entsteht häufig beim Lesen von Programmen, wenn bei der Aufnahme die Cassette klemmte und jetzt bei der Wiedergabe 'leiert'.
Der dritte mögliche Fehler ist der READ ERROR D. Dieser Fehler dürfte nur in den seltensten Fällen auftreten, da er signalisiert, daß der gelesene Block länger als die zulässigen 2048 Bytes ist. Das kann aber nur auftreten, wenn der Anwender beim Speichern in die Header-Information größere Werte als erlaubt einträgt.

Sicher kennen Sie den Basic-Befehl **'SPEED WRITE par'**. Je nach verwendetem Parameter werden Daten mit durchschnittlich 1000 oder 2000 Baud auf Cassette gespeichert. Damit ist aber noch nicht die obere Grenze der Geschwindigkeit erreicht. Durch Verwendung einer Betriebssystem-Routine läßt sich jede Baudrate zwischen 700 Baud und etwa 3600 Baud einstellen. Die benötigte Routine hat ihren Einsprung auf der Adresse &BC68. Sie erwartet in zwei Registern Parameter und stellt entsprechend die Schreibgeschindigkeit ein.
Ein Wert wird im HL-Registerpaar übergeben und bestimmt die Baudrate. Die Formel zur Bestimmung dieses Wertes lautet:

Baudrate = 333333/halbe Länge eines Null-Bits

Bei 1000 Baud ergibt sich daraus eine Zeit von 666

Microsekunden für ein Null-Bit, ein Eins-Bit ist genau doppelt so lang.

Die im Recorder verwendete Elektronik hat aber eine Besonderheit. Werden abwechselnd Null- und Eins-Bits gelesen, so versucht die Elektronik die Zeitunterschiede auszugleichen. Dadurch werden Eins-Bits kürzer, Null-Bits aber erscheinen als längere Impulse als nach der Aufzeichnung zu erwarten wären. Aus diesem Grund muß eine Vorkompensation durchgeführt werden, die Null-Bits werden kürzer aufgezeichnet, Eins-Bits werden mit geringfügig längeren Zeiten aufgezeichnet. Diese für die Vorkompensation benötigten Zeiten werden im Akku der Routine übergeben.

Für Versuche zur Bestimmung der höchsten, halbwegs zuverlässigen Schreibgeschwindigkeit genügt es, im Akku einen Wert von 10 zu übergeben. Um mit 3600 Baud aufzuzeichen, muß die folgende Routine einmal aktiviert werden:

```
LD HL,93
LD A,10
CALL &BC68
RET
```

Diese wenigen Bytes können leicht mit den folgenden Zeilen in den Speicher gelegt werden:

```
10 MEMORY HIMEM - 10
20 FOR I = 1 TO 9
30 READ X : POKE HIMEM + I,X
40 NEXT I
50 CALL HIMEM+1
60 DATA &21,&5D,&00,&3E,&0A,&CD,&68,&BC,&C9
```

Varieren Sie ruhig etwas mit den Werten in HL und Akku (der zweite und der fünfte Wert in der Data-Zeile), um die maximale Aufzeichnungsfrequenz zu bestimmen. Diese ist vom verwendeten Cassettenmaterial abhängig. Aber auch die

Gleichlaufeigenschaften Ihres Recorders spielen eine nicht unerhebliche Rolle bei der Zuverlässigkeit hoher Aufnahmegeschwindigkeiten.

Werden die Werte zu klein gewählt, dann kann der CPC die geforderten Zeiten nicht mehr einhalten, als Ergebnis erhalten Sie die Fehlermeldung WRITE ERROR A.

Zum Schluß noch ein Tip, der geleichermaßen für Disketten- wie Cassettenbetrieb gilt:
Sie werden sicher beim Abspeichern sehr langer Programme mit vielen Variablen bemerkt haben, daß es bis zu 15 Minuten dauern kann, bis die Daten oder das Programm gespeichert sind. Das liegt an der Tatsache, daß der CPC zum Speichern einen Bereich von 2K für die zu übertragenden Blocks benötigt. Dieser Buffer wird an der oberen Speichergrenze angelegt. Ist dieser Bereich jedoch mit Variablen belegt, dann werden diese Variablen in einen anderen Speicherbereich copiert. Dieser Vorgang ist vergleichbar mit der viel gefürchteten Garbage Collection, die immer dann auftritt, wenn im Speicher für Zeichenketten und Arrays kein ausreichender Platz vorhanden ist.
Die durch die Variablenverschiebung auftretende Wartezeit kann man aber deutlich reduzieren, indem zu Beginn des jeweiligen Programms dieser 2k-Puffer bereits angelegt und geschützt wird. Ein möglicher Programmanfang könnte folgendermaßen aussehen:

```
10 OPENOUT "DUMMY"
20 MEMORY HIMEM-1
30 CLOSEOUT
40
50 'REST DES PROGRAMMS
```

Dieser Vorgang ist natürlich nur sinnvoll, wenn Sie in dem entsprechenden Programm auch mit Dateien arbeiten. Ist das nicht der Fall, dann können Sie auf die gezeigten Programmzeilen verzichten und vor dem gewünschten Abspeichern den Befehl CLEAR eingeben. Dadurch werden alle

zuvor definierten Variablen gelöscht und das Anlegen des Cassettenpuffers geht ohne nennenswerte Zeit vonstatten.

1.10.5 Die Centronics-Druckerschnittstelle

Man findet an jedem Computer etwas, das man für verbesserungswürdig hält. Beim CPC ist das ohne Frage die Druckerschnittstelle. Obwohl viele Schwächen und Fehler des CPC 464 bei seinen Nachfolgern 664 und 6128 ausgemerzt wurden, hat man beim ärgerlichsten Schwachpunkt, der Druckerschnittstelle, keine Änderung vorgenommen. Auch bei diesen Rechnern ist weiterhin nur eine 7-Bit-Schnittstelle eingebaut. Da aber die meisten Drucker, sogar der von Schneider zum CPC angebotene, einen 8-Bit-Eingang haben, sind viele Kommandos und Möglichkeiten dieser Drucker nur über Umwege oder überhaupt nicht zu erreichen.

Aber betrachten wir zunächst den hardwaremäßigen Aufbau der Schnittstelle.
In der Hauptsache besteht die Schnittstelle aus einem 8-fachen Latch 74LS273. Die acht einzelnen Latches arbeiten wie Flip-Flops, die an den Eingängen liegende Information wird mit einer High-Low-Flanke am Takt-Eingang Pin 11 gespeichert und steht bis zu einem RESET oder einer Neuprogrammierung an den Ausgängen zur Verfügung, unabhängig von sich ändernden Eingangssignalen.
Das Taktsignal, dessen High-Low-Flanke das Speichern der Eingangswerte bewirkt, wird mit einem OR-Gatter erzeugt. Der Ausgang des Gatters wird dann Low, wenn beide Eingänge Low sind.
Auch der Druckeranschluß wird über die Portadressierung angesprochen. Aus diesem Grund liegt das Signal IOWR* an einem Eingang des OR-Gatters, am anderen Eingang liegt die Adressleitung A12.
Ähnlich der Adressierung der anderen Peripherie-Bausteinen ist die Dekodierung also sehr unvollständig. Entsprechend müssen alle Adressleitungen, die nicht für die Dekodierung benötigt werden, High sein, um Kollisionen mit anderen verwendeten Portadressen zu vermeiden. Damit ergibt sich eine effektive Portadresse von &EFxx.

Die Eingänge des Drucker-Latch sind mit dem Prozessor-Datenbus verbunden. Die Ausgänge liegen am Druckeranschluß. Nur das Bit 7 wird über ein als Inverter benutztes NAND-Gatter an den Centronics-Port gelegt. Dies Bit stellt das für den Drucker benötigte Strobe-Signal dar. Normalerweise ist dies Signal High. Will der Rechner aber ein Zeichen an den Drucker schicken, so legt er das zu übertragende Byte auf die Datenleitungen und kurz darauf das Strobe-Signal auf Low. Damit wird das zu übertragende Byte von Drucker akzeptiert.
Voraussetzung dafür ist allerdings, daß das Signal Busy des Druckers Low ist. Der Zustand des Busy-Signals wird vom Bit 6 des 8255-Ports B abgefragt.
Wie aber kann das Strobe-Signal erzeugt werden? Nichts einfacher als das.
Jedes zu übertragende Byte wird zuerst mit &7F verUNDet. Damit ist das oberste Bit des Bytes mit Sicherheit gelöscht. Dieses Byte wird per OUT-Befehl auf den Printer-Port ausgegeben.
Jetzt liegen die zu übertragenden Bits bereits am Drucker an, das Strobe-Signal ist über den Inverter aber immer noch High. Darum wird anschließend mit OR &80 das Bit 7 des auszugebenden Wertes gesetzt und ebenfalls auf den Printer-Port ausgegeben. An dem zu übertragenden Wert hat sich nichts geändert, nur das Strobe-Signal ist durch den Inverter Low geworden.
Dieses Signal muß aber auch wieder High werden, darum wird mit UND das oberste Bit wieder gelöscht und das Byte noch einmal ausgegeben. Damit ist ein Byte vom Rechner zum Drucker geschickt worden.

Von BASIC aus ist die Ausgabe auf den Drucker kein Problem. Aber auch in Maschinensprache muß nicht der ganze 'Kram' selbst geschrieben werden. Es gibt mehrere Routinen, die einem einigen Programmieraufwand abnehmen.
Da ist zunächst die Routine mit Einsprung bei &BD2B. Über diese Routine kann man ein Zeichen auf den Drucker ausgeben.

Das jeweilige Zeichen muß sich dabei im Akku befinden. Zusätzlich prüft diese Routine, ob der Drucker 'Busy' ist. Meldet sich der Drucker innerhalb von 0.4 Sekunden nicht, dann kehrt die Routine mit gelöschtem Carry-Flag zurück. Dann muß ein neuer Versuch mit demselben Zeichen gestartet werden. Diese Routine wird auch vom Basic-Interpreter verwendet. Bei geglückter Übertragung ist das Carry gesetzt. Daraufhin kann das nächste Zeichen gesendet werden.

Eine weitere Routine hat ihren Einsprung drei Bytes weiter (&BD2E). Diese Routine kann genutzt werden, um den Zustand des Druckers zu prüfen. Ist kein Drucker angeschlossen oder meldet der Drucker 'Busy', kann also momentan keine Zeichen entgegen nehmen, dann kehrt diese Routine mit gesetztem Carry zurück, andernfalls ist das Carry gelöscht.

Die dritte verwertbare Routine (&BD31) erledigt alle Vorgänge, um ein Zeichen auf dem Drucker auszugeben. Dabei muß aber der Programmierer vorher prüfen, ob der Drucker empfangsbereit ist und das gewünschte Zeichen im Akku übergeben. Wird die Überprüfung des Zustands versäumt, dann geht das Zeichen eventuell ins 'Leere'.

Wie diese Routinen einsetzbar sind, finden Sie etwas später in diesem Buch. Dort zeigen wir am Beispiel einer Text- und einer Grafik-Hardcopy den Einsatz dieser und anderer Routinen, die den Gebrauch verdeutlicht.

Aber noch eine Besonderheit gilt es bei der vorhandenen Beschaltung des Centronics-Anschlusses zu berücksichtigen.
Die Kontaktbelegung des Druckerports verleitet geradezu, sich die benötigten Steckverbindungen und ein Stück Flachbandkabel zu besorgen und sich ein solches Kabel selbst zu fertigen. Handelt es sich bei den Verbindern dazu noch um sogenannte Quetschverbinder, dann haben auch handwerklich ungeschickte CPC-Besitzer ein solches Kabel in 5 bis 10 Minuten selbst zusammengestellt. Damit lassen sich dann alle Drucker mit Centronics-Eingang am CPC verwenden.

Aber beim ersten Probelauf gibt es eine große Überraschung. Der Drucker geht erstaunlich großzügig mit dem Papier um. Nach jeder gedruckten Zeile wird erst einmal eine Leerzeile eingeschoben.

Ursache ist folgendes:
Der CPC fügt an das Ende jeder Druckzeile die Zeichenfolge CR/LF, also die Befehlsfolge für Wagenrücklauf und Zeilenvorschub. Dadurch wird das Papier eine Zeile weiter transportiert. Zusätzlich und ohne ersichtlichen Grund ist aber noch der Pin 14 des Centronics-Anschlusses des CPC mit Masse verbunden. Das bewirkt bei den meisten Druckern einen weiteren Zeilenvorschub, so daß immer eine Leerzeile produziert wird.

Abhilfe schafft in diesem Fall die Unterbrechung der Leitung zum Pin 14 des Druckers. Nach dem Auftrennen dieser Leitung und evtl. nötigem Einstellen von Schaltern im Drucker wie z.B bei Epson sollte aber alles in Ordnung sein.

1.10.6 Der Joystickanschluß

Hauptsächlich wird der Joystickanschluß sicher so genutzt, daß er seinem Namen auch gerecht wird: als Eingang zur Abfrage eines Joysticks. Über sieben der verfügbaren 9 Anschlüsse lassen sich aber auch andere Tasten oder Schalter abfragen. Bei entsprechender Programmierung und bei Verzicht auf Interrupt und Tastatur-Abfrage könnten diese sieben Anschlüsse sogar als Ausgang betrieben werden. Die Joystickanschlüsse sind ja mit dem bidirektionalen Port des Sound Chip verbunden und könnten bei den erwähnten Einschränkungen aus als Ausgang arbeiten. Allerdings ist für Ausgabezwecke der Centronics-Port sicher einfacher zu handhaben.

Wie schon im Kapitel 1.10.1 beschrieben, werden die Joysticks als Tasten der Tastatur angesehen. Aus diesem Grund sind die benötigten sieben Eingänge des Sound Chip-Ports auf die Joystickbuchse gelegt. Zusätzlich sind noch zwei Ausgänge des bei der Tastaturbeschreibung erwähnten BCD-Dezimal-Dekoders auf die Buchse gelegt.

Jede fünfzigstel Sekunde wird einmal komplett die Tastatur abgefragt. Dabei wird auch der Zustand der Joysticks abgefragt. Für Basic-Programme steht der Zustand der Joysticks mit der JOY(nummer)-Funktion zur Verfügung. Auch mit INKEY könnte der Zustand der Joysticks einfach bestimmt werden. Aber auch für Assembler-Fans gibt es die Möglichkeit, den Zustand der Joysticks einfach zu bestimmen. Die System-Routine &BB24 liefert im HL-Doppelregister den aktuellen Zustand in Bits verschlüsselt zurück. Im H-Register und im Akku erhält man beim Aufruf dieser Routine den Zustand des Joystick 0, das L-Register gilt für Joystick 1. Die Verschlüsselung der Joysticktasten erfolgt nach dem gleichen Schema wie bei der JOY (x)-Funktion, Bit 0 ist gesetzt für vorwärts, Bit 1 für rückwärts, Bit 2 für links und so weiter.

1.10.7 Der Expansion-Connector

Diese Schnittstelle ist die universellste des CPC. An diesem 50-poligen Leiterplattenverbinder befinden sich neben allen Signalen des Prozessors auch noch verschiedene Steuersignale. Hier werden alle Erweiterungen des Systems angeschlossen.

Die Bedeutung der Signale 3 bis 39 ist ja bereits aus der Beschreibung des Prozessors bekannt. Darum wollen wir uns hier auf die verbleibenden Anschlüsse beschränken.

Am Pin 1 steht das Soundsignal noch einmal zur Verfügung. Allerdings ist dies Signal nur Mono, alle drei Kanäle werden direkt hier zugeführt.

Pin 2 und Pin 49 sind mit der Masse der Stromversorgung verbunden.

Eine Besonderheit ist das Signal BUS-RESET* am Pin 40. Durch Low-legen dieses Signals wird ein Reset des Systems durchgeführt.

Leider löscht der CPC beim Reset den ganzen Speicher. Darum ist dieses Signal als 'Notbremse' für abgestürzte Maschinenprogramme genau so wirkungsvoll wie das aus- und wieder einschalten.

Am Pin 41 steht das eigentliche RESET-Signal für externe Erweiterungen zur Verfügung. Beachten Sie aber, das nicht alle Bausteine mit diesem RESET-Signal versorgt werden können. Der 8255 z.B. benötigt dieses Signal invertiert.

Sehr interessant sind die beiden Signale ROMEN* und ROMDIS. Das an Pin 42 des Expansion Connectors liegende ROMEN* signalisiert bei Low-Pegel einen Zugriff auf das eingebaute 32K-Rom. Dieser Zugriff kann aber durch High-Pegel am Pin 43, ROMDIS unterbunden werden. Dadurch kann das gesamte eingebaute Rom durch externe Roms oder Eproms ersetzt werden.
Bei entsprechender Dekodierung der Adressleitungen können aber auch nur bestimmte Bereiche im eingebauten Rom ausgeblendet und ersetzt werden.

Eine ähnliche Funktion haben die beiden Signale RAMRD* und RAMDIS für Lesezugriffe auf das interne Ram. Diese an den Pins 43 und 44 liegenden Signale können genutzt werden, um z.B. bestimmte Ram-Bereiche gegen Roms oder auch Rams auszutauschen.
Die Ansteuerung externer Rams ist allerdings beim CPC nicht so ganz einfach. Hauptsächliche Schwierigkeit ist die Tatsache, daß das WR*-Signal für die internen Rams nicht vom Prozessor, sondern vom Gate Array produziert wird. Dieser Schreibimpuls kann leider durch keinen Programmtrick verhindert werden, so daß ein Schreibzugriff auf ein externes Ram auch immer das interne Ram adressiert und beschreibt. Die im CPC 6128 eingebauten 64 KByte können auch nur durch den beschriebenen Einsatz des PAL-Bausteins in der gewünschten Weise angesprochen werden.

Das am Pin 46 verfügbare Signal CURSOR wird bei entsprechender Programmierung vom Video-Controller

geliefert. Der CRTC verfügt ja über die Möglichkeit des Hardware-Cursors. Je nach Programmierung erscheint an diesem Ausgang ein Rechteck-Signal mit einer Frequenz von etwa 1.5 oder 3 Hertz. Aber auch ständige Low- und High-Pegel an diesem Anschluß sind programmierbar.
Nach dem Einschalten des CPC liegt hier ein ständiger Low-Pegel.

Der LPEN-Eingang (Light Pen) an Pin 47 ist direkt mit dem Light Pen-Eingang des CRTC verbunden. Dieses IC verfügt über alle nötigen Register zum Betrieb des Lightpen.
Allerdings wird die Nutzung des Light Pen besonders bei hochauflösender Grafik im CPC sehr schwierig realisierbar, da der Video-Controller zwar die MA-Adresse der momentanen Light Pen-Position liefert, aber keine Angaben über die aktuelle RA-Adresse macht. Durch den speziellen Aufbau des Video-Ram ist diese Angabe aber nötig, wenn auf dem Bildschirm mit dem Light Pen gezeichnet werden soll.

Der Eingang Pin 48 trägt die Bezeichnung EXP* und ist mit dem 8255-Port B Bit 4 verbunden. Eine externe Erweiterung kann diesen Anschluß an Masse legen und sich auf diese Weise dem Betriebssystem bemerkbar machen.

Das letzte zu erwähnende Signal am Pin 50 ist das Taktsignal des Prozessors. Dieses Signal mit einer Frequenz von 4 MHz kann als Taktsignal für externe Peripherie-ICs verwendet werden.

2 DAS BETRIEBSSYSTEM

Hinter diesem, für den Uneingeweihten nichtssagenden Namen verbirgt sich der Dreh- und Angelpunkt des ganzen Rechners. Hier ist praktisch das Stellwerk, in dem die Weichen zur Verbindung von Anwenderprogramm und Hardware gestellt werden.
Auch der Basic-Interpreter ist in diesem Zusammenhang als Programm zu sehen, welches via Betriebssystem auf die Hardware des Rechners zugreift. Dieser verbindenden Funktion wegen haben wir das Kapitel auch in die Mitte des Buches gerückt.
Der Aufbau des Betriebssystems ist logisch klar untergliedert in sog. Packs, von denen jedes einen speziellen Aufgabenbereich hat. Das beginnt auf der untersten Ebene beim **MACHINE PACK**, welches am hardwarenächsten ist und z.B. den Printer-Port, die Sound-Register usw. bedient, dann weiter über das **SCREEN PACK**, das den Bildschirm handhabt und seinerseits vom **TEXT PACK** und **GRAPHICS PACK** aufgerufen wird.
Beim genauen Hinsehen fällt auf, daß jedes Pack streng in sich geschlossen ist, und die Kommunikation mit anderen Packs über genau definierte Schnittstellen erfolgt. Darber hinaus verfügt jedes Pack über einen eigenen Ram-Bereich als Arbeitsspeicher. Der Ansprung von Routinen erfolgt in der Regel über Vektoren im Ram, selten über die direkte Rom-Adresse.
Das legt die Vermutung nahe, daß das Betriebssystem, vermutlich der Kürze der zur Verfügung stehenden Zeit wegen, von mehreren Programmierern geschrieben wurde, jeder für ein oder mehrere Packs zuständig, nachdem man sich nur über die Schnittstellen geeinigt hat.
Wie dem auch gewesen sein mag, jedenfalls eröffnet dieser klare Aufbau und der Zugriff bis in den kleinsten Winkel über Vektoren dem Anwendungsprogrammierer ungeahnte und bisher nicht gekannte Aspekte.
Als Beispiel sei nur die Möglichkeit genannt, einen Treiber für einen echten acht Bit Drucker (wie auch immer dieser hardwaremäßig angeschlossen sein mag) zu schreiben und

diesen durch Verbiegen nur des Vektors **MC WAIT PRINTER** dem gesamten System zugänglich zu machen.
Dieser Tip soll Ihnen auch als Warnung dienen: **Bedienen Sie sich ruhig der Routinen des Betriebssystems, aber nur über die Vektoren!** Es könnte ja bereits jemand anders (Rom Cartridge) einige Vektoren verstellt haben, um bestimmte Funktionen erst einmal über eigene Routinen zu leiten. Sie sähen mit Ihrem Druckertreiber ja auch ganz schön alt aus, wenn die Ausgabe für die File-# 8 direkt über $07F2 laufen würde und nicht über $BDF1.

Mit der Zeit werden Sie schon selber darauf kommen, daß eigene Programme mit minimalem Aufwand geschrieben werden können, wenn man sich nur fleißig der Vektoren bedient. Absolut neu ist, daß sogar die Arithmetikroutinen von Basic über diesen Mechanismus führen, was einerseits dazu dienen kann, eigene Berechnungen dort ausführen zu lassen, andererseits, um eigene Programme dort einzuhängen, weil Sie vielleicht eine höhere Genauigkeit wünschen, usw.

Da wir Ihnen jetzt so viel von Vektoren vorgeschwärmt haben, beginnen wir im nächsten Kapitel gleich damit.

2.1 Die Betriebssystem-Vektoren

Auf den folgenden Seiten stellen wir Ihnen die RAM-Adressen vor, über die Sie Routinen im Betriebssystem anspringen oder die Sie bei Bedarf verändern können, um bestimmte Funktionen über eigene Programme zu führen. Teils handelt es sich hier um komplette Routinen, die ins RAM kopiert wurden und in die Sie mitten hinein springen können, teils um RST 1 oder RST 5, gefolgt von der Inline-Adress, die ins ROM verweist.

Im Anhang finden Sie eine Auflistung der Routinen im ROM, die beim schnellen Wiederfinden der gesamten Routine behilflich sein soll.

2.1.1 Die Betriebssystem-Vektoren des CPC 664

B900	KL U ROM ENABLE Aktuelles oberes ROM einschalten.
B903	KL U ROM DISABLE Oberes ROM ausschalten.
B906	KL L ROM ENABLE Unteres ROM einschalten.
B909	KL L ROM DISABLE Unteres ROM ausschalten.
B90C	KL ROM RESTORE Alte ROM-Konfiguration wiederherstellen.
B90F	KL ROM SELECT Ein bestimmtes oberes ROM auswählen.
B912	KL CURR SELECTION Welches obere ROM ist an?
B915	KL PROBE ROM ROM untersuchen.
B918	KL ROM DESELECT Alte obere ROM-Konfiguration wiederherstellen.
B91B	KL LDIR LDIR bei blockierten ROMs.
B91E	KL LDDR LDDR bei blockierten ROMs.
B921	KL POLL SYNCHRONOUS Gibt es einen Event mit höherer Priorität als die des laufenden?

B941 RST 7 INTERRUPT ENTRY CONT'D Einsprung für Hardware-Interrupts.
B978 KL EXT INTERRUPT ENTRY
B984 KL LOW PCHL CONT'D Sprung ins untere ROM oder RAM.
B98A RST 1 LOW JUMP CONT'D Aufruf einer Routine im Betriebssystem oder im darunterliegenden RAM.
B9B9 KL FAR PCHL CONT'D
B9C1 KL FAR ICALL CONT'D
B9C7 RST 3 LOW FAR CALL CONT'D Es kann eine Routine irgendwo im RAM oder ROM aufgerufen werden.
BA17 KL SIDE PCHL CONT'D
BA1D RST 2 LOW SIDE CALL CONT'D Dient zum Aufruf einer Routine im Expansion-ROM.
BA35 RST 5 FIRM JUMP CONT'D Ermöglicht das Springen zu einer Routine im Betriebssystem.
BA51 KL L ROM ENABLE CONT'D Unteres ROM einschalten.
BA58 KL L ROM DISABLE CONT'D Unteres ROM ausschalten.
BA5F KL U ROM ENABLE CONT'D Oberes ROM einschalten.
BA66 KL U ROM DISABLE CONT'D Oberes ROM ausschalten.
BA70 KL ROM RESTORE CONT'D Alte ROM-Konfiguration wiederherstellen.
BA79 KL ROM SELECT CONT'D Ein bestimmtes oberes ROM auswählen.
BA7E KL PROBE ROM CONT'D ROM untersuchen.
BA87 KL ROM DESELECT CONT'D Alte obere ROM-Konfiguration wiederherstellen.
BA9D KL CURR SELECTION CONT'D Welches obere ROM ist an?
BAA1 KL LDIR CONT'D LDIR bei blockierten ROMs.
BAA7 KL LDDR CONT'D LDDR bei blockierten ROMs.
BAAD KL ROM OFF & KONFIG. SAVE
BAC6 RST 4 RAM LAM CONT'D RAM-Inhalt lesen, unabhängig vom ROM-Zustand.

BAD7 KL RAM LAM (IX) Entspricht ld a,(ix).

BB00 KM INITIALISE Vollständige Initialisierung der Tastaturverwaltung.
BB03 KM RESET Rücksetzen der Tastaturverwaltung.
BB06 KM WAIT CHAR Auf ein Zeichen von der Tastatur warten.
BB09 KM READ CHAR Zeichen von der Tastatur holen, falls ein Zeichen vorhanden ist.
BB0C KM CHAR RETURN Zeichen im Tastaturpuffer für den nächsten Zugriff hinterlegen.
BB0F KM SET EXPAND Erweiterungsstring einrichten.
BB12 KM GET EXPAND Zeichen vom Erweiterungsstring holen.
BB15 KM EXP BUFFER Speicher für Erweiterungsstring zuweisen.
BB18 KM WAIT KEY Auf Tastendruck warten.
BB1B KM READ KEY Tastennummer holen, falls eine Taste gedrückt wurde.
BB1E KM TEST KEY Ist eine Taste gedrückt worden?
BB21 KM GET STATE Shift-Status holen.
BB24 KM GET JOYSTICK Der momentane Zustand der Joysticks wird abgefragt.
BB27 KM SET TRANSLATE Eintrag in die Tastaturtabelle vornehmen (1. Ebene).
BB2A KM GET TRANSLATE Eintrag aus der Tastaturtabelle holen (1. Ebene).
BB2D KM SET SHIFT Eintrag in die Tastaturtabelle vornehmen (2. Ebene).
BB30 KM GET SHIFT Eintrag aus der Tastaturtabelle holen (2. Ebene).
BB33 KM SET CONTROL Eintrag in die Tastaturtabelle vornehmen (3. Ebene).
BB36 KM GET CONTROL Eintrag aus der Tastaturtabelle holen (3. Ebene).
BB39 KM SET REPEAT Wiederholungsfunktion für eine bestimmte Taste setzen.
BB3C KM GET REPEAT Wiederholungsfunktion für eine bestimmte Taste gesetzt?

BB3F KM SET DELAY Tastenwiederholungseinsatz und -geschwindigkeit setzen.
BB42 KM GET DELAY Parameter für Tastenwiederholungseinsatz und -geschwindigkeit holen.
BB45 KM ARM BREAK Break-Taste zulassen.
BB48 KM DISARM BREAK Break-Taste verriegeln.
BB4B KM BREAK EVENT Routinen bei dem Drücken der Break-Taste ausführen.

BB4E TXT INITIALISE Vollständige Initialisierung des Text-Packs.
BB51 TXT RESET Rücksetzen des Text-Packs.
BB54 TXT VDU ENABLE Es können Zeichen auf den Bildschirm geschrieben werden.
BB57 TXT VDU DISABLE Zeichendarstellung unterbinden.
BB5A TXT OUTPUT (Steuer-)Zeichen darstellen oder ausführen.
BB5D TXT WR CHAR Zeichen darstellen.
BB60 TXT RD CHAR Zeichen vom Bildschirm lesen.
BB63 TXT SET GRAPHIC Darstellung von Steuerzeichen ein- oder ausschalten.
BB66 TXT WIN ENABLE Größe des lfd. Textfensters festlegen.
BB69 TXT GET WINDOW Welche Größe hat das lfd. Textfenster?
BB6C TXT CLEAR WINDOW Lfd. Textfenster löschen.
BB6F TXT SET COLUMN Horizontale Position des Cursors setzen.
BB72 TXT SET ROW Vertikale Position des Cursors setzen.
BB75 TXT SET CURSOR Cursor positionieren.
BB78 TXT GET CURSOR Abfrage der momentanen Cursorposition.
BB7B TXT CUR ENABLE Cursor erlauben (Anwenderprogramm).
BB7E TXT CUR DISABLE Cursor verriegeln (Anwenderprogramm).
BB81 TXT CUR ON Cursor erlauben (Betriebssystem).

BB84 TXT CUR OFF Cursor verriegeln (Betriebssystem, höhere Priorität als BB7B TXT CUR ENABLE/BB7E TXT CUR DISABLE).
BB87 TXT VALIDATE Cursor innerhalb des Textfensters?
BB8A TXT PLACE/REMOVE CURSOR Cursor auf den Bildschirm setzen/Cursor vom Bildschirm nehmen.
BB8D TXT PLACE/REMOVE CURSOR Cursor auf den Bildschirm setzen/Cursor vom Bildschirm nehmen.
BB90 TXT SET PEN Vordergrundfarbe setzen.
BB93 TXT GET PEN Welche Vordergrundfarbe?
BB96 TXT SET PAPER Hintergrundfarbe setzen.
BB99 TXT GET PAPER Welche Hintergrundfarbe?
BB9C TXT INVERSE Lfd. Vorder- und Hintergrundfarbe vertauschen.
BB9F TXT SET BACK Transparentmodus ein/aus.
BBA2 TXT GET BACK Welcher Transparentmodus?
BBA5 TXT GET MATRIX Adresse des Punktmusters eines Zeichens holen.
BBA8 TXT SET MATRIX Adresse des (vom Anwender definierten) Punktmusters eines bestimmten Zeichen setzen.
BBAB TXT SET M TABLE Startadresse und erstes Zeichen einer vom Anwender definierten Punktmatrix setzen.
BBAE TXT GET M TABLE Startadresse und erstes Zeichen einer Anwendermatrix?
BBB1 TXT GET CONTROLS Adresse der Steuerzeichen-Sprungtabelle holen.
BBB4 TXT STR SELECT Textfenster wählen.
BBB7 TXT SWAP STREAMS Die Parameter (Farben, Fenstergrenzen usw.) zweier Textfenster werden miteinander vertauscht.

BBBA GRA INITIALISE Vollständige Initialisierung des Graphik-Packs.
BBBD GRA RESET Zurücksetzen des Graphik-Packs.
BBC0 GRA MOVE ABSOLUTE Bewegung zu einer absoluten Position.
BBC3 GRA MOVE RELATIVE Bewegung relativ zur momentanen Position.

BBC6 GRA ASK CURSOR Wo ist der lfd. Graphikcursor?
BBC9 GRA SET ORIGIN Ursprung der Anwender-Koordinaten setzen.
BBCC GRA GET ORIGIN Ursprung der Anwender-Koordinaten holen.
BBCF GRA WIN WIDTH Linke und rechte Begrenzung des Graphikfensters setzen.
BBD2 GRA WIN HEIGHT Obere und untere Begrenzung des Graphikfensters setzen.
BBD5 GRA GET W WIDTH Linke und rechte Begrenzung des Graphikfensters?
BBD8 GRA GET W HEIGHT Obere und untere Begrenzung des Graphikfensters?
BBDB GRA CLEAR WINDOW Graphikfenster löschen.
BBDE GRA SET PEN Schreibfarbe setzen.
BBE1 GRA GET PEN Welche Schreibfarbe?
BBE4 GRA SET PAPER Hintergrundfarbe setzen.
BBE7 GRA GET PAPER Welche Hintergrundfarbe?
BBEA GRA PLOT ABSOLUTE Graphikpunkt setzen (absolut).
BBED GRA PLOT RELATIVE Graphikpunkt setzen (relativ zum lfd. Cursor).
BBF0 GRA TEST ABSOLUTE Punkt gesetzt (absolut)?
BBF3 GRA TEST RELATIVE Punkt gesetzt (relativ zum lfd. Cursor)?
BBF6 GRA LINE ABSOLUTE Linie von der lfd. zur absoluten Position ziehen.
BBF9 GRA LINE RELATIVE Linie von der lfd. bis zur rel. Distanz ziehen.
BBFC GRA WR CHAR Ein Zeichen an der lfd. Graphikcursorposition schreiben.

BBFF SCR INITIALISE Initialisierung des Screen-Packs.
BC02 SCR RESET Rücksetzen des Screen-Packs.
BC05 SCR SET OFFSET Startadresse des ersten Zeichens relativ zur Basisadresse des Video-RAMs setzen.
BC08 SCR SET BASE Basisadresse des Video-RAMs setzen.
BC0B SCR GET LOCATION Lfd. Bildschirmstart? (Basis+Offset).
BC0E SCR SET MODE Bildschirmmodus setzen.

BC11 SCR GET MODE Bildschirmmodus holen und testen.
BC14 SCR CLEAR Bildschirm löschen.
BC17 SCR CHAR LIMITS Größtmögliche Zeilen- und Spaltenzahl des Bildschirms holen (abhängig vom Modus).
BC1A SCR CHAR POSITION Phys. Koordinaten in Bildschirmposition umsetzen.
BC1D SCR DOT POSITION Bildschirmposition für einen Pixel ermitteln.
BC20 SCR NEXT BYTE Eine gegebene Bildschirmadresse um eine Zeichenposition weiterrechnen.
BC23 SCR PREV BYTE Bildschirmadresse um eine Position zurückrechnen.
BC26 SCR NEXT LINE Bildschirmadresse um eine Zeile weiterrechnen.
BC29 SCR PREV LINE Bildschirmadresse um eine Zeile zurückrechnen.
BC2C SCR INK ENCODE Eine Ink in eine verschlüsselte Form bringen.
BC2F SCR INK DECODE Eine Ink entschlüsseln.
BC32 SCR SET INK Farbe(n) einer Ink-# zuordnen.
BC35 SCR GET INK Farbe(n) einer Ink-# ?
BC38 SCR SET BORDER Rahmenfarbe(n) setzen.
BC3B SCR GET BORDER Rahmenfarbe(n)?
BC3E SCR SET FLASHING Blinkzeiten setzen.
BC41 SCR GET FLASHING Blinkzeiten?
BC44 SCR FILL BOX Vorgegebenes Fenster mit einer Farbe füllen (Positionen zeichenbezogen, Mode-abhängig).
BC47 SCR FLOOD BOX Vorgegebenes Fenster mit einer Farbe füllen (Positionen sind Bildschirmadressen, Mode-unabhängig).
BC4A SCR CHAR INVERT Bei einem Zeichen Vorder- und Hintergrundfarbe vertauschen.
BC4D SCR HW ROLL Bildschirm eine Zeile auf oder ab (hardwaremäßig).
BC50 SCR SW ROLL Bildschirm eine Zeile auf oder ab (softwaremäßig).
BC53 SCR UNPACK Zeichenmatrix vergrößern (für Mode 0/1).

BC56	SCR REPACK	Zeichenmatrix wieder auf Originalform stauchen.
BC59	SCR ACCESS	Steuerzeichen sichtbar/unsichtbar setzen.
BC5C	SCR PIXELS	Punkt auf dem Bildschirm setzen.
BC5F	SCR HORIZONTAL	Horizontale Linie ziehen.
BC62	SCR VERTICAL	Vertikale Linie ziehen.
BC65	CAS INITIALISE	Kassettenmanager initialisieren.
BC68	CAS SET SPEED	Schreibgeschwindigkeit setzen.
BC6B	CAS NOISY	Kassettenmeldungen ein/aus.
BC6E	CAS START MOTOR	Kassettenmotor starten.
BC71	CAS STOP MOTOR	Kassettenmotor stoppen.
BC74	CAS RESTORE MOTOR	Alten Motorzustand wiederherstellen.
BC77	CAS IN OPEN	Eröffnen einer Eingabedatei.
BC7A	CAS IN CLOSE	Ordnungsgemäßes Schließen einer Eingabedatei.
BC7D	CAS IN ABANDON	Eingabedatei sofort schließen.
BC80	CAS IN CHAR	Zeichen lesen (aus dem Puffer).
BC83	CAS IN DIRECT	Gesamte Datei in den Speicher ziehen.
BC86	CAS RETURN	Zuletzt gelesenes Zeichen wieder zurück in den Puffer.
BC89	CAS TEST EOF	Dateiende?
BC8C	CAS OUT OPEN	Eröffnen einer Ausgabedatei.
BC8F	CAS OUT CLOSE	Ordnungsgemäßes Schließen einer Ausgabedatei.
BC92	CAS OUT ABANDON	Ausgabedatei sofort schließen.
BC95	CAS OUT CHAR	Zeichen schreiben (in den Puffer).
BC98	CAS OUT DIRECT	Definierten Speicherbereich auf Kassette schreiben (nicht über den Puffer).
BC9B	CAS CATALOG	Gib einen Katalog der Kassette auf dem Bildschirm aus.
BC9E	CAS WRITE	Block schreiben.
BCA1	CAS READ	Block lesen.
BCA4	CAS CHECK	Block auf dem Band mit Speicherinhalt vergleichen.
BCA7	SOUND RESET	Rücksetzen des Sound-Packs.
BCAA	SOUND QUEUE	Ton an die Warteschlange anhängen.

BCAD	SOUND CHECK	Noch Platz in der Warteschlange?
BCB0	SOUND ARM EVENT	Eventblock für den Fall 'scharf machen', daß in der Warteschlange ein Platz frei wird.
BCB3	SOUND RELEASE	Töne erlauben.
BCB6	SOUND HOLD	Töne sofort anhalten.
BCB9	SOUND CONTINUE	Zuvor angehaltene Töne weiter bearbeiten.
BCBC	SOUND AMPL ENVELOPE	Lautstärkehüllkurve einrichten.
BCBF	SOUND TONE ENVELOPE	Tonhüllkurve einrichten.
BCC2	SOUND A ADRESS	Adresse einer Lautstärkehüllkurve holen.
BCC5	SOUND T ADRESS	Adresse einer Tonhüllkurve holen.
BCC8	KL CHOKE OFF	Kernel rücksetzen.
BCCB	KL ROM WALK	Irgendwelche ROM-Erweiterungen?
BCCE	KL INIT BACK	ROM-Erweiterung einhängen.
BCD1	KL LOG EXT	Residente Erweiterung einhängen.
BCD4	KL FIND COMMAND	Befehl in allen eingehängten Speicherbereichen suchen.
BCD7	KL NEW FRAME FLY	Eventblock einrichten und einhängen.
BCDA	KL ADD FRAME FLY	Eventblock einhängen.
BCDD	KL DEL FRAME FLY	Eventblock aushängen.
BCE0	KL NEW FAST TICKER	Wie BCD7.
BCE3	KL ADD FAST TICKER	Wie BCDA.
BCE6	KL DEL FAST TICKER	Wie BCDD.
BCE9	KL ADD TICKER	Tickerblock einhängen.
BCEC	KL DEL TICKER	Tickerblock aushängen.
BCEF	KL INIT EVENT	Eventblock einrichten.
BCF2	KL EVENT	Eventblock 'kicken'.
BCF5	KL SYNC RESET	Sync Pending Queue löschen.
BCF8	KL DEL SYNCHRONOUS	Bestimmten Block aus der Pending Queue löschen.
BCFB	KL NEXT SYNC	Der nächste bitte.
BCFE	KL DO SYNC	Eventroutine ausführen.
BD01	KL DONE SYNC	Eventroutine fertig.

BD04 KL EVENT DISABLE Sperren der normalen gleichzeitigen Ereignisse. Eilige gleichzeitige Ereignisse werden nicht gesperrt.
BD07 KL EVENT ENABLE Normale gleichzeitige Ereignisse zulassen.
BD0A KL DISARM EVENT Eventblock verriegeln (Zähler negativ).
BD0D KL TIME PLEASE Wieviel Zeit ist abgelaufen?
BD10 KL TIME SET Setzen der Zeit auf einen vorgegebenen Wert.

BD13 MC BOOT PROGRAM Setzt das Betriebssystem zurück und übergibt die Steuerung einer Routine in (hl).
BD16 MC START PROGRAM Initialisierung des Systems und Aufruf eines Programms.
BD19 MC WAIT FLYBACK Strahlrücklauf abwarten.
BD1C MC SET MODE Bildschirmmodus setzen.
BD1F MC SCREEN OFFSET Bildschirmoffset setzen.
BD22 MC CLEAR INKS Bildschirmrand und Inks auf eine Farbe setzen.
BD25 MC SET INKS Farben für alle Inks setzen.
BD28 MC RESET PRINTER Rücksetzen des indirekten Verzweigungspunktes für den Drucker.
BD2B MC PRINT CHAR Zeichen drucken wenn möglich.
BD2E MC BUSY PRINTER Drucker noch beschäftigt?
BD31 MC SEND PRINTER Zeichen drucken (warten, bis es möglich ist).
BD34 MC SOUND REGISTER Sound Controller mit Daten versorgen.

BD37 JUMP RESTORE Alle Sprungvektoren initialisieren.

BD3A KM SET STATE
BD3D KM PUFFER ENTLEEREN
BD40 TXT LFD. CURSOR FLAG NACH AKKU
BD43 GRA NN
BD46 GRA PARAM RETTEN
BD49 GRA MASK PARAM RETTEN
BD4C GRA MASK PARAM RETTEN

BD4F GRA KOORD. KONVERTIEREN Logische in physikalische Koordinaten.
BD52 GRA FILL Fillroutine
BD55 SCR VERÄNDERUNG SCREEN START
BD58 MC ZEICHENZUORDNUNG

Die folgenden Vektoren werden vom BASIC benutzt.

BD5B EDIT

BD5E FLO VARIABLE VON (DE) NACH (HL) KOPIEREN
BD61 FLO INTEGER NACH FLIESSKOMMA
BD64 FLO 4-BYTE-WERT NACH FLO
BD67 FLO FLO NACH INT
BD6A FLO FLO NACH INT
BD6D FLO FIX
BD70 FLO INT
BD73 FLO
BD76 FLO ZAHL MIT 10^A MULTIPLIZIEREN
BD79 FLO ADDITION
BD7C FLO RND
BD7F FLO SUBTRAKTION
BD82 FLO MULTIPLIKATION
BD85 FLO DIVISION
BD88 FLO LETZTEN RND-WERT HOLEN
BD8B FLO VERGLEICH
BD8E FLO VORZEICHENWECHSEL
BD91 FLO SGN
BD94 FLO DEG/RAD
BD97 FLO PI
BD9A FLO SQR
BD9D FLO POTENZIERUNG
BDA0 FLO LOG
BDA3 FLO LOG10
BDA6 FLO EXP
BDA9 FLO SIN

BDAC FLO COS
BDAF FLO TAN
BDB2 FLO ATN
BDB5 FLO 4-BYTE-WERT NACH FLO
BDB8 FLO RND INIT
BDBB FLO SET RND SEED

Hier beginnen die sogenannten **INDIRECTIONS**. Das sind Sprünge ins Betriebssystem, die nicht global versorgt werden, sondern individuell von jedem Pack, wenn dessen **RESET** oder **INITIALISE** durchlaufen wird.

BDCD TXT DRAW/UNDRAW CURSOR Setzen/Löschen des Cursors.
BDD0 TXT DRAW/UNDRAW CURSOR Setzen/Löschen des Cursors
BDD3 TXT WRITE CHAR Ein Zeichen auf den Bildschirm schreiben.
BDD6 TXT UNWRITE CHAR Ein Zeichen vom Bildschirm lesen.
BDD9 TXT OUT ACTION Ausgabe eines Zeichens auf dem Bildschirm oder Ausführung eines Steuercodes.

BDDC GRA PLOT Stelle einen Punkt auf dem Bildschirm dar.
BDDF GRA TEST Gib die Ink der momentanen Graphik-Position.
BDE2 GRA LINE Zeichnen einer Linie.

BDE5 SCR READ Lesen eines Pixels und entschlüsseln der Ink.
BDE8 SCR WRITE Pixel(s) schreiben.
BDEB SCR CLEAR Löschen des Bildschirms.

BDEE KM TEST BREAK ESC, SHIFT und CTRL führen zum Rücksetzen des gesamten Systems.

BDF1 MC WAIT PRINTER Sende eine Zeichen an den Drucker; wenn dieser nicht bereit ist, warte eine Zeitperiode.

BDF4 KM UPDATE KEY STATE MAP

2.1.2 Die Betriebssystem-Vektoren des CPC 6128

B900 KL U ROM ENABLE Aktuelles oberes ROM einschalten.
B903 KL U ROM DISABLE Oberes ROM ausschalten.
B906 KL L ROM ENABLE Unteres ROM einschalten.
B909 KL L ROM DISABLE Unteres ROM ausschalten.
B90C KL ROM RESTORE Alte ROM-Konfiguration wiederherstellen.
B90F KL ROM SELECT Ein bestimmtes oberes ROM auswählen.
B912 KL CURR SELECTION Welches obere ROM ist an?
B915 KL PROBE ROM ROM untersuchen.
B918 KL ROM DESELECT Alte obere ROM-Konfiguration wiederherstellen.
B91B KL LDIR LDIR bei blockierten ROMs.
B91E KL LDDR LDDR bei blockierten ROMs.

B921 KL POLL SYNCHRONOUS Gibt es einen Event mit höherer Priorität als die des laufenden?
B941 RST 7 INTERRUPT ENTRY CONT'D Einsprung für Hardware-Interrupts.
B978 KL EXT INTERRUPT ENTRY
B984 KL LOW PCHL CONT'D Sprung ins untere ROM oder RAM.
47498 B98A RST 1 LOW JUMP CONT'D Aufruf einer Routine im Betriebssystem oder im darunterliegenden RAM.
B9B9 KL FAR PCHL CONT'D
B9C1 KL FAR ICALL CONT'D
B9C7 RST 3 LOW FAR CALL CONT'D Es kann eine Routine irgendwo im RAM oder ROM aufgerufen werden.
BA17 KL SIDE PCHL CONT'D
BA1D RST 2 LOW SIDE CALL CONT'D Dient zum Aufruf einer Routine im Expansion-ROM.
BA35 RST 5 FIRM JUMP CONT'D Ermöglicht das Springen zu einer Routine im Betriebssystem.

BA51 KL L ROM ENABLE CONT'D Unteres ROM einschalten.
BA58 KL L ROM DISABLE CONT'D Unteres ROM ausschalten.
BA5F KL U ROM ENABLE CONT'D Oberes ROM einschalten.
BA66 KL U ROM DISABLE CONT'D Oberes ROM ausschalten.
BA70 KL ROM RESTORE CONT'D Alte ROM-Konfiguration wiederherstellen.
BA79 KL ROM SELECT CONT'D Ein bestimmtes oberes ROM auswählen.
BA7E KL PROBE ROM CONT'D ROM untersuchen.
BA87 KL ROM DESELECT CONT'D Alte obere ROM-Konfiguration wiederherstellen.
BA9D KL CURR SELECTION CONT'D Welches obere ROM ist an?
BAA1 KL LDIR CONT'D LDIR bei blockierten ROMs.
BAA7 KL LDDR CONT'D LDDR bei blockierten ROMs.
BAAD KL ROM OFF & KONFIG. SAVE
BAC6 RST 4 RAM LAM CONT'D RAM-Inhalt lesen, unabhängig vom ROM-Zustand.
BAD7 KL RAM LAM (IX) Entspricht ld a,(ix).

BB00 KM INITIALISE Vollständige Initialisierung der Tastaturverwaltung.
BB03 KM RESET Rücksetzen der Tastaturverwaltung.
BB06 KM WAIT CHAR Auf ein Zeichen von der Tastatur warten.
BB09 KM READ CHAR Zeichen von der Tastatur holen, falls ein Zeichen vorhanden ist.
BB0C KM CHAR RETURN Zeichen im Tastaturpuffer für den nächsten Zugriff hinterlegen.
BB0F KM SET EXPAND Erweiterungsstring einrichten.
BB12 KM GET EXPAND Zeichen vom Erweiterungsstring holen.
BB15 KM EXP BUFFER Speicher für Erweiterungsstring zuweisen.
BB18 KM WAIT KEY Auf Tastendruck warten.

BB1B KM READ KEY Tastennummer holen, falls eine Taste gedrückt wurde.

BB1E KM TEST KEY Ist eine Taste gedrückt worden?

BB21 KM GET STATE Shift-Status holen.

BB24 KM GET JOYSTICK Der momentane Zustand der Joysticks wird abgefragt.

BB27 KM SET TRANSLATE Eintrag in die Tastaturtabelle vornehmen (1. Ebene).

BB2A KM GET TRANSLATE Eintrag aus der Tastaturtabelle holen (1. Ebene).

BB2D KM SET SHIFT Eintrag in die Tastaturtabelle vornehmen (2. Ebene).

BB30 KM GET SHIFT Eintrag aus der Tastaturtabelle holen (2. Ebene).

BB33 KM SET CONTROL Eintrag in die Tastaturtabelle vornehmen (3. Ebene).

BB36 KM GET CONTROL Eintrag aus der Tastaturtabelle holen (3. Ebene).

BB39 KM SET REPEAT Wiederholungsfunktion für eine bestimmte Taste setzen.

BB3C KM GET REPEAT Wiederholungsfunktion für eine bestimmte Taste gesetzt?

BB3F KM SET DELAY Tastenwiederholungseinsatz und -geschwindigkeit setzen.

BB42 KM GET DELAY Parameter für Tastenwiederholungseinsatz und -geschwindigkeit holen.

BB45 KM ARM BREAK Break-Taste zulassen.

BB48 KM DISARM BREAK Break-Taste verriegeln.

BB4B KM BREAK EVENT Routinen bei dem Drücken der Break-Taste ausführen.

BB4E TXT INITIALISE Vollständige Initialisierung des Text-Packs.

BB51 TXT RESET Rücksetzen des Text-Packs.

BB54 TXT VDU ENABLE Es können Zeichen auf den Bildschirm geschrieben werden.

BB57 TXT VDU DISABLE Zeichendarstellung unterbinden.

BB5A TXT OUTPUT (Steuer-)Zeichen darstellen oder ausführen.

47965 BB5D TXT WR CHAR Zeichen darstellen.

BB60 TXT RD CHAR Zeichen vom Bildschirm lesen.

BB63 TXT SET GRAPHIC Darstellung von Steuerzeichen ein- oder ausschalten.

BB66 TXT WIN ENABLE Größe des lfd. Textfensters festlegen.

BB69 TXT GET WINDOW Welche Größe hat das lfd. Textfenster?

47980 BB6C TXT CLEAR WINDOW Lfd. Textfenster löschen.

47983 BB6F TXT SET COLUMN Horizontale Position des Cursors setzen.

47986 BB72 TXT SET ROW Vertikale Position des Cursors setzen.

BB75 TXT SET CURSOR Cursor positionieren.

BB78 TXT GET CURSOR Abfrage der momentanen Cursorposition.

BB7B TXT CUR ENABLE Cursor erlauben (Anwenderprogramm).

BB7E TXT CUR DISABLE Cursor verriegeln (Anwenderprogramm).

BB81 TXT CUR ON Cursor erlauben (Betriebssystem).

BB84 TXT CUR OFF Cursor verriegeln (Betriebssystem, höhere Priorität als BB7B TXT CUR ENABLE/BB7E TXT CUR DISABLE).

BB87 TXT VALIDATE Cursor innerhalb des Textfensters?

BB8A TXT PLACE/REMOVE CURSOR Cursor auf den Bildschirm setzen/Cursor vom Bildschirm nehmen.

BB8D TXT PLACE/REMOVE CURSOR Cursor auf den Bildschirm setzen/Cursor vom Bildschirm nehmen.

BB90 TXT SET PEN Vordergrundfarbe setzen.

BB93 TXT GET PEN Welche Vordergrundfarbe?

BB96 TXT SET PAPER Hintergrundfarbe setzen.

BB99 TXT GET PAPER Welche Hintergrundfarbe?

BB9C TXT INVERSE Lfd. Vorder- und Hintergrundfarbe vertauschen.

BB9F TXT SET BACK Transparentmodus ein/aus.

BBA2 TXT GET BACK Welcher Transparentmodus?

BBA5 TXT GET MATRIX Adresse des Punktmusters eines Zeichens holen.

BBA8 TXT SET MATRIX Adresse des (vom Anwender definierten) Punktmusters eines bestimmten Zeichen setzen.
BBAB TXT SET M TABLE Startadresse und erstes Zeichen einer vom Anwender definierten Punktmatrix setzen.
BBAE TXT GET M TABLE Startadresse und erstes Zeichen einer Anwendermatrix?
BBB1 TXT GET CONTROLS Adresse der Steuerzeichen-Sprungtabelle holen.
BBB4 TXT STR SELECT Textfenster wählen.
BBB7 TXT SWAP STREAMS Die Parameter (Farben, Fenstergrenzen usw.) zweier Textfenster werden miteinander vertauscht.

BBBA GRA INITIALISE Vollständige Initialisierung des Graphik-Packs.
BBBD GRA RESET Zurücksetzen des Graphik-Packs.
BBC0 GRA MOVE ABSOLUTE Bewegung zu einer absoluten Position.
BBC3 GRA MOVE RELATIVE Bewegung relativ zur momentanen Position.
BBC6 GRA ASK CURSOR Wo ist der lfd. Graphikcursor?
BBC9 GRA SET ORIGIN Ursprung der Anwender-Koordinaten setzen.
BBCC GRA GET ORIGIN Ursprung der Anwender-Koordinaten holen.
BBCF GRA WIN WIDTH Linke und rechte Begrenzung des Graphikfensters setzen.
BBD2 GRA WIN HEIGHT Obere und untere Begrenzung des Graphikfensters setzen.
BBD5 GRA GET W WIDTH Linke und rechte Begrenzung des Graphikfensters?
BBD8 GRA GET W HEIGHT Obere und untere Begrenzung des Graphikfensters?
BBDB GRA CLEAR WINDOW Graphikfenster löschen.
BBDE GRA SET PEN Schreibfarbe setzen.
BBE1 GRA GET PEN Welche Schreibfarbe?
BBE4 GRA SET PAPER Hintergrundfarbe setzen.
BBE7 GRA GET PAPER Welche Hintergrundfarbe?

BBEA GRA PLOT ABSOLUTE Graphikpunkt setzen (absolut).
BBED GRA PLOT RELATIVE Graphikpunkt setzen (relativ zum lfd. Cursor).
BBF0 GRA TEST ABSOLUTE Punkt gesetzt (absolut)?
BBF3 GRA TEST RELATIVE Punkt gesetzt (relativ zum lfd. Cursor)?
BBF6 GRA LINE ABSOLUTE Linie von der lfd. zur absoluten Position ziehen.
BBF9 GRA LINE RELATIVE Linie von der lfd. bis zur rel. Distanz ziehen.
BBFC GRA WR CHAR Ein Zeichen an der lfd. Graphikcursorposition schreiben.

BBFF SCR INITIALISE Initialisierung des Screen-Packs.
BC02 SCR RESET Rücksetzen des Screen-Packs.
BC05 SCR SET OFFSET Startadresse des ersten Zeichens relativ zur Basisadresse des Video-RAMs setzen.
BC08 SCR SET BASE Basisadresse des Video-RAMs setzen.
BC0B SCR GET LOCATION Lfd. Bildschirmstart? (Basis+Offset).
48142 BC0E SCR SET MODE Bildschirmmodus setzen.
BC11 SCR GET MODE Bildschirmmodus holen und testen.
48148 BC14 SCR CLEAR Bildschirm löschen.
BC17 SCR CHAR LIMITS Größtmögliche Zeilen- und Spaltenzahl des Bildschirms holen (abhängig vom Modus).
48154 BC1A SCR CHAR POSITION Phys. Koordinaten in Bildschirmposition umsetzen.
BC1D SCR DOT POSITION Bildschirmposition für einen Pixel ermitteln.
BC20 SCR NEXT BYTE Eine gegebene Bildschirmadresse um eine Zeichenposition weiterrechnen.
BC23 SCR PREV BYTE Bildschirmadresse um eine Position zurückrechnen.
BC26 SCR NEXT LINE Bildschirmadresse um eine Zeile weiterrechnen.
BC29 SCR PREV LINE Bildschirmadresse um eine Zeile zurückrechnen.
BC2C SCR INK ENCODE Eine Ink in eine verschlüsselte Form bringen.

BC2F SCR INK DECODE Eine Ink entschlüsseln.
BC32 SCR SET INK Farbe(n) einer Ink-# zuordnen.
BC35 SCR GET INK Farbe(n) einer Ink-# ?
BC38 SCR SET BORDER Rahmenfarbe(n) setzen.
BC3B SCR GET BORDER Rahmenfarbe(n)?
BC3E SCR SET FLASHING Blinkzeiten setzen.
BC41 SCR GET FLASHING Blinkzeiten?
BC44 SCR FILL BOX Vorgegebenes Fenster mit einer Farbe füllen (Positionen zeichenbezogen, Mode-abhängig).
BC47 SCR FLOOD BOX Vorgegebenes Fenster mit einer Farbe füllen (Positionen sind Bildschirmadressen, Mode-unabhängig).
BC4A SCR CHAR INVERT Bei einem Zeichen Vorder- und Hintergrundfarbe vertauschen.
BC4D SCR HW ROLL Bildschirm eine Zeile auf oder ab (hardwaremäßig).
BC50 SCR SW ROLL Bildschirm eine Zeile auf oder ab (softwaremäßig).
BC53 SCR UNPACK Zeichenmatrix vergrößern (für Mode 0/1).
BC56 SCR REPACK Zeichenmatrix wieder auf Originalform stauchen.
BC59 SCR ACCESS Steuerzeichen sichtbar/unsichtbar setzen.
BC5C SCR PIXELS Punkt auf dem Bildschirm setzen.
BC5F SCR HORIZONTAL Horizontale Linie ziehen.
BC62 SCR VERTICAL Vertikale Linie ziehen.

BC65 CAS INITIALISE Kassettenmanager initialisieren.
BC68 CAS SET SPEED Schreibgeschwindigkeit setzen.
BC6B CAS NOISY Kassettenmeldungen ein/aus.
BC6E CAS START MOTOR Kassettenmotor starten.
BC71 CAS STOP MOTOR Kassettenmotor stoppen.
BC74 CAS RESTORE MOTOR Alten Motorzustand wiederherstellen.
BC77 CAS IN OPEN Eröffnen einer Eingabedatei.
BC7A CAS IN CLOSE Ordnungsgemäßes Schließen einer Eingabedatei.
BC7D CAS IN ABANDON Eingabedatei sofort schließen.
BC80 CAS IN CHAR Zeichen lesen (aus dem Puffer).

BC83 CAS IN DIRECT Gesamte Datei in den Speicher ziehen.
BC86 CAS RETURN Zuletzt gelesenes Zeichen wieder zurück in den Puffer.
BC89 CAS TEST EOF Dateiende?
BC8C CAS OUT OPEN Eröffnen einer Ausgabedatei.
BC8F CAS OUT CLOSE Ordnungsgemäßes Schließen einer Ausgabedatei.
BC92 CAS OUT ABANDON Ausgabedatei sofort schließen.
BC95 CAS OUT CHAR Zeichen schreiben (in den Puffer).
BC98 CAS OUT DIRECT Definierten Speicherbereich auf Kassette schreiben (nicht über den Puffer).
BC9B CAS CATALOG Gib einen Katalog der Kassette auf dem Bildschirm aus.
BC9E CAS WRITE Block schreiben.
BCA1 CAS READ Block lesen.
BCA4 CAS CHECK Block auf dem Band mit Speicherinhalt vergleichen.

BCA7 SOUND RESET Rücksetzen des Sound-Packs.
BCAA SOUND QUEUE Ton an die Warteschlange anhängen.
BCAD SOUND CHECK Noch Platz in der Warteschlange?
BCB0 SOUND ARM EVENT Eventblock für den Fall 'scharf machen', daß in der Warteschlange ein Platz frei wird.
BCB3 SOUND RELEASE Töne erlauben.
BCB6 SOUND HOLD Töne sofort anhalten.
BCB9 SOUND CONTINUE Zuvor angehaltene Töne weiter bearbeiten.
BCBC SOUND AMPL ENVELOPE Lautstärkehüllkurve einrichten.
BCBF SOUND TONE ENVELOPE Tonhüllkurve einrichten.
BCC2 SOUND A ADRESS Adresse einer Lautstärkehüllkurve holen.
BCC5 SOUND T ADRESS Adresse einer Tonhüllkurve holen.

BCC8 KL CHOKE OFF Kernel rücksetzen.
BCCB KL ROM WALK Irgendwelche ROM-Erweiterungen?
BCCE KL INIT BACK ROM-Erweiterung einhängen.
BCD1 KL LOG EXT Residente Erweiterung einhängen.

BCD4 KL FIND COMMAND Befehl in allen eingehängten Speicherbereichen suchen.

BCD7 KL NEW FRAME FLY Eventblock einrichten und einhängen.

BCDA KL ADD FRAME FLY Eventblock einhängen.

BCDD KL DEL FRAME FLY Eventblock aushängen.

BCE0 KL NEW FAST TICKER Wie BCD7.

BCE3 KL ADD FAST TICKER Wie BCDA.

BCE6 KL DEL FAST TICKER Wie BCDD.

BCE9 KL ADD TICKER Tickerblock einhängen.

BCEC KL DEL TICKER Tickerblock aushängen.

BCEF KL INIT EVENT Eventblock einrichten.

BCF2 KL EVENT Eventblock 'kicken'.

BCF5 KL SYNC RESET Sync Pending Queue löschen.

BCF8 KL DEL SYNCHRONOUS Bestimmten Block aus der Pending Queue löschen.

BCFB KL NEXT SYNC Der nächste bitte.

BCFE KL DO SYNC Eventroutine ausführen.

BD01 KL DONE SYNC Eventroutine fertig.

BD04 KL EVENT DISABLE Sperren der normalen gleichzeitigen Ereignisse. Eilige gleichzeitige Ereignisse werden nicht gesperrt.

BD07 KL EVENT ENABLE Normale gleichzeitige Ereignisse zulassen.

BD0A KL DISARM EVENT Eventblock verriegeln (Zähler negativ).

BD0D KL TIME PLEASE Wieviel Zeit ist abgelaufen?

BD10 KL TIME SET Setzen der Zeit auf einen vorgegebenen Wert.

BD13 MC BOOT PROGRAM Setzt das Betriebssystem zurück und übergibt die Steuerung einer Routine in (hl).

BD16 MC START PROGRAM Initialisierung des Systems und Aufruf eines Programms.

BD19 MC WAIT FLYBACK Strahlrücklauf abwarten.

BD1C MC SET MODE Bildschirmmodus setzen.

BD1F MC SCREEN OFFSET Bildschirmoffset setzen.

BD22 MC CLEAR INKS Bildschirmrand und Inks auf eine Farbe setzen.

BD25	MC SET INKS Farben für alle Inks setzen.
BD28	MC RESET PRINTER Rücksetzen des indirekten Verzweigungspunktes für den Drucker.
BD2B	MC PRINT CHAR Zeichen drucken wenn möglich.
BD2E	MC BUSY PRINTER Drucker noch beschäftigt?
BD31	MC SEND PRINTER Zeichen drucken (warten, bis es möglich ist).
BD34	MC SOUND REGISTER Sound Controller mit Daten versorgen.
BD37	JUMP RESTORE Alle Sprungvektoren initialisieren.
BD3A	KM SET STATE
BD3D	KM PUFFER ENTLEEREN
BD40	TXT LFD. CURSOR FLAG NACH AKKU
BD43	GRA NN
BD46	GRA PARAM RETTEN
BD49	GRA MASK PARAM RETTEN
BD4C	GRA MASK PARAM RETTEN
BD4F	GRA KOORD. KONVERTIEREN Logische in physikalische Koordinaten.
BD52	GRA FILL
BD55	SCR VERÄNDERUNG SCREEN START
BD58	MC ZEICHENZUORDNUNG
BD5B	KL RAM-KONFIGURATION SETZEN

BASIC-Vektoren

BD5E	EDIT
BD61	FLO VARIABLE VON (DE) NACH (HL) KOPIEREN
BD64	FLO INTEGER NACH FLIESSKOMMA
BD67	FLO 4-BYTE-WERT NACH FLO
BD6A	FLO FLO NACH INT
BD6D	FLO FLO NACH INT
BD70	FLO FIX

BD73	FLO INT
BD76	FLO
BD79	FLO ZAHL MIT 10^A MULTIPLIZIEREN
BD7C	FLO ADDITION
BD7F	FLO RND
BD82	FLO SUBTRAKTION
BD85	FLO MULTIPLIKATION
BD88	FLO DIVISION
BD8B	FLO LETZTEN RND-WERT HOLEN
BD8E	FLO VERGLEICH
BD91	FLO VORZEICHENWECHSEL
BD94	FLO SGN
BD97	FLO DEG/RAD
BD9A	FLO PI
BD9D	FLO SQR
BDA0	FLO POTENZIERUNG
BDA3	FLO LOG
BDA6	FLO LOG10
BDA9	FLO EXP
BDAC	FLO SIN
BDAF	FLO COS
BDB2	FLO TAN
BDB5	FLO ATN
BDB8	FLO 4-BYTE-WERT NACH FLO
BDBB	FLO RND INIT
BDBE	FLO SET RND SEED

INDIRECTIONS

BDCD	TXT DRAW/UNDRAW CURSOR Setzen/Löschen des Cursors.
BDD0	TXT DRAW/UNDRAW CURSOR Setzen/Löschen des Cursors
BDD3	TXT WRITE CHAR Ein Zeichen auf den Bildschirm schreiben.

BDD6 TXT UNWRITE CHAR Ein Zeichen vom Bildschirm lesen.

BDD9 TXT OUT ACTION Ausgabe eines Zeichens auf dem Bildschirm oder Ausführung eines Steuercodes.

BDDC GRA PLOT Stelle einen Punkt auf dem Bildschirm dar.

BDDF GRA TEST Gib die Ink der momentanen Graphik-Position.

BDE2 GRA LINE Zeichnen einer Linie.

BDE5 SCR READ Lesen eines Pixels und entschlüsseln der Ink.

BDE8 SCR WRITE Pixel(s) schreiben.

BDEB SCR CLEAR Löschen des Bildschirms.

BDEE KM TEST BREAK ESC, SHIFT und CTRL führen zum Rücksetzen des gesamten Systems.

BDF1 MC WAIT PRINTER Sende eine Zeichen an den Drucker; wenn dieser nicht bereit ist, warte eine Zeitperiode.

BDF4 KM UPDATE KEY STATE MAP

2.2 Das Betriebssystem-RAM

Hier finden Sie eine Auflistung des Betriebssystem-RAMs, soweit wir die Bedeutungen der einzelnen Adressen herausgefunden haben.

Voraussetzung für ihre Benutzung ist allerdings, daß Sie sich über die Wirkung von Manipulationen vorher genau im klaren sind. Anderenfalls könnte es passieren, daß Sie nicht nur bedeutungslose Pointer oder Flags zerstören, sondern wesentlich schwerwiegendere Dinge wie beispielsweise die Sprungtabelle des **TEXT SCREENs.**

2.2.1 Das Betriebssystem-RAM des CPC 664

B82D	KL Start Int Pending Queue
B831	KL div. Flags f. Int. Rout.
B832	KL sp save
B8B4	KL Timer low
B8B6	KL Timer high
B8B8	KL Timerflag
B8B9	KL Start Frame Fly Chain
B8BB	KL Start Fast Ticker Chain
B8BD	KL Start Ticker Chain
B8BF	KL Count for Ticker
B8C0	KL Start Sync Pending Queue
B8C2	KL Priorität lfd. Event
B8C3	KL auszuführender Befehl
B8D5	KL lfd. Exp.-ROM
B8D6	KL Einsprung lfd. ROM
B8D8	KL lfd. ROM-Konfiguration

B7C3	SCR curr. Screen Mode
B7C4	SCR Position in einer Zeile

B7C6	SCR High Byte Screen Start
B7C7	SCR Write Indirection
B7D2	SCR Flash Periods
B7D3	SCR Flash Period 1. Farbe
B7D4	SCR Farbspeicher 2. Farben
B7E5	SCR Farbspeicher 1. Farben
B7F6	SCR Flag lfd. Farbsatz
B7F8	SCR curr. Flash Period
B7F9	SCR Event Block: Set Inks

B6B5	TXT lfd. Bildschirmfenster
B6B6	TXT Start Params Fenster 0
B726	TXT lfd. Cursor Pos. (Row, Col)
B728	TXT Fenst. Flag (0=ges. Bildsch.)
B729	TXT lfd. Fenster oben
B72A	TXT lfd. Fenster links
B72B	TXT lfd. Fenster unten
B72C	TXT lfd. Fenster rechts
B72D	TXT lfd. Roll Count
B72E	TXT lfd. Cursor Flag
B72F	TXT lfd. Pen
B730	TXT lfd. Paper
B731	TXT lfd. Background Mode
B733	TXT Graph Char Write Mode (0=disable)
B734	TXT 1. Zeichen User Matrix
B736	TXT Adr. User Matrix
B758	TXT Zeichenzähler Control Buffer
B759	TXT Start Control Buffer
B763	TXT Sprungtabelle Steuerzeichen

B693	GRA X Origin
B695	GRA Y Origin
B697	GRA lfd. X Koord.
B699	GRA lfd. Y Koord.
B69B	GRA X Koord. GRA Fenster links
B69D	GRA X Koord. GRA Fenster rechts
B69F	GRA Y Koord. GRA Fenster oben

B6A1	GRA Y Koord. GRA Fenster unten
B6A3	GRA Pen
B6A4	GRA Paper
B6A5	GRA Rechenpuffer X Koordinaten
B6A7	GRA Rechenpuffer Y Koordinaten

B628	KM Exp. String Pointer
B62A	KM Put Back Buffer
B62B	KM Adr. Start Exp. Buffer
B62D	KM Adr. Ende Exp. Buffer
B62F	KM Adr. Start Freier Exp. Buffer
B631	KM Shift Lock State
B632	KM Caps Lock State
B633	KM Delay
B635	KM Key State Map
B637	KM Key 16...23
B62B	KM Joystick 1
B63E	KM Joystick 0
B63F	KM während Scan gedrückte Keys
B649	KM Multihit Kontr. zu B63F
B657	KM Break Event Block
B68B	KM Adr. Key Translation Table
B68D	KM Adr. Key SHIFT Table
B68F	KM Adr. Key CTRL Table
B691	KM Adr. der Repeat Tabelle

B1ED	SOUND alte Sound Akt. (nach. HOLD)
B1EE	SOUND lfd. Sound Aktivität
B1F8	SOUND Params Kanal A
B237	SOUND Params Kanal B
B276	SOUND Params Kanal C
B2A6	SOUND Lautstärke Hüllkurven
B396	SOUND Tonhüllkurven

B118	CAS Cass. Message Flag
B11A	CAS Input Buffer Status

B11B	CAS Adr. Start Input Buffer
B11D	CAS Pointer Input Buffer
B11F	CAS File Header Input
B15F	CAS Output Buffer Status
B160	CAS Adr. Start Output Buffer
B162	CAS Pointer Output Buffer
B164	CAS File Header Output
B1E9	CAS Cass. Speed
B115	EDIT Insert Flag

2.2.2 Das Betriebssystem-RAM des CPC 6128

B82D	KL Start Int Pending Queue
B831	KL div. Flags f. Int. Rout.
B832	KL sp save
B8B4	KL Timer low
B8B5	KL lfd. RAM-Konfiguration
B8B6	KL Timer high
B8B8	KL Timerflag
B8B9	KL Start Frame Fly Chain
B8BB	KL Start Fast Ticker Chain
B8BD	KL Start Ticker Chain
B8BF	KL Count for Ticker
B8C0	KL Start Sync Pending Queue
B8C2	KL Priorität lfd. Event
B8C3	KL auszuführender Befehl
B8D6	KL lfd. Exp.-ROM
B8D7	KL Einsprung lfd. ROM
B8D9	KL lfd. ROM-Konfiguration

B7C3	SCR curr. Screen Mode
B7C4	SCR Position in einer Zeile
B7C6	SCR High Byte Screen Start
B7C7	SCR Write Indirection
B7D2	SCR Flash Periods
B7D3	SCR Flash Period 1. Farbe
B7D4	SCR Farbspeicher 2. Farben
B7E5	SCR Farbspeicher 1. Farben
B7F6	SCR Flag lfd. Farbsatz
B7F8	SCR curr. Flash Period
B7F9	SCR Event Block: Set Inks

B6B5	TXT lfd. Bildschirmfenster
B6B6	TXT Start Params Fenster 0
B726	TXT lfd. Cursor Pos. (Row, Col)

B728	TXT Fenst. Flag (0=ges. Bildsch.)
B729	TXT lfd. Fenster oben
B72A	TXT lfd. Fenster links
B72B	TXT lfd. Fenster unten
B72C	TXT lfd. Fenster rechts
B72D	TXT lfd. Roll Count
B72E	TXT lfd. Cursor Flag
B72F	TXT lfd. Pen
B730	TXT lfd. Paper
B731	TXT lfd. Background Mode
B733	TXT Graph Char Write Mode (0=disable)
B734	TXT 1. Zeichen User Matrix
B736	TXT Adr. User Matrix
B758	TXT Zeichenzähler Control Buffer
B759	TXT Start Control Buffer
B763	TXT Sprungtabelle Steuerzeichen

B693	GRA X Origin
B695	GRA Y Origin
B697	GRA lfd. X Koord.
B699	GRA lfd. Y Koord.
B69B	GRA X Koord. GRA Fenster links
B69D	GRA X Koord. GRA Fenster rechts
B69F	GRA Y Koord. GRA Fenster oben
B6A1	GRA Y Koord. GRA Fenster unten
B6A3	GRA Pen
B6A4	GRA Paper
B6A5	GRA Rechenpuffer X Koordinaten
B6A7	GRA Rechenpuffer Y Koordinaten

B628	KM Exp. String Pointer
B62A	KM Put Back Buffer
B62B	KM Adr. Start Exp. Buffer
B62D	KM Adr. Ende Exp. Buffer
B62F	KM Adr. Start Freier Exp. Buffer
B631	KM Shift Lock State
B632	KM Caps Lock State

B633	KM Delay
B635	KM Key State Map
B637	KM Key 16...23
B62B	KM Joystick 1
B63E	KM Joystick 0
B63F	KM während Scan gedrückte Keys
B649	KM Multihit Kontr. zu B63F
B657	KM Break Event Block
B68B	KM Adr. Key Translation Table
B68D	KM Adr. Key SHIFT Table
B68F	KM Adr. Key CTRL Table
B691	KM Adr. der Repeat Tabelle

B1ED	SOUND alte Sound Akt. (nach. HOLD)
B1EE	SOUND lfd. Sound Aktivität
B1F8	SOUND Params Kanal A
B237	SOUND Params Kanal B
B276	SOUND Params Kanal C
B2A6	SOUND Lautstärke Hüllkurven
B396	SOUND Tonhüllkurven

B118	CAS Cass. Message Flag
B11A	CAS Input Buffer Status
B11B	CAS Adr. Start Input Buffer
B11D	CAS Pointer Input Buffer
B11F	CAS File Header Input
B15F	CAS Output Buffer Status
B160	CAS Adr. Start Output Buffer
B162	CAS Pointer Output Buffer
B164	CAS File Header Output
B1E9	CAS Cass. Speed

B115	EDIT Insert Flag

2.3 Nutzung von Betriebssystemroutinen

Der CPC enthält mehrere hundert z.T. sehr sinnvolle und für den Programmierer gut zu benutzende Routinen oder Funktionen. So gibt es solche Routinen zur Abfrage der Tastatur, zum Ausgeben eines Zeichens auf dem Bildschirm, zum Verwalten der Windows oder zum Betrieb des Druckers.
Trotz der Fülle der im Betriebssystem vorhandenen Funktionen gibt es aber Dinge, die der CPC von Haus aus nicht kann. So fehlt z.B. die Möglichkeit, den Inhalt des Bildschirms, Text oder Grafik, auf einen angeschlossenen Drucker auszugeben.

Diese '**Hardcopy**' genannte Möglichkeit wollen wir Ihnen an zwei Beispielen zeigen. Im ersten Beispiel geht es dabei um eine ausschließliche Texthardcopy, die mit jedem angeschlossenen Drucker funktioniert, die zweite Hardcopy-Routine ermöglicht den Ausdruck aller Zeichen einschließlich der CPC-Grafikzeichen. Auch in hochauflösender Grafik erstellte Bilder lassen sich mit dieser Routine ausdrucken. Als Drucker wurde der NLQ 401 gewählt. Dieser preiswerte Drucker ist, was seinen Vorrat an Steuerzeichen angeht, erstaunlich kompatibel mit den Epson-Druckern MX/RX/FX. Dadurch laufen die beiden Programme auch auf den erwähnten Epson-Druckern (und allen sonstigen Kompatiblen) ohne Anpassung.

Am Ende dieses Kapitels haben Sie nicht nur zwei schnelle Hardcopy-Routinen, sondern auch einen Einblick in die Nutzung einiger Betriebssystem-Routinen.

Um den Bildschirminhalt auf einen angeschlossenen Drucker auszugeben, muß man die Zeichen zeilenweise vom Bildschirm lesen und ausgeben. Durch den speziellen Aufbau des Video-Ram kann man leider nicht direkt die Zeichen auslesen.

Über den 'Umweg' einer Betriebssystem-Routine kann aber das Zeichen an der momentanen Cursor-Position bestimmt werden. Diese Routine (**TXT RD CHAR, &BB60**) übergibt das Zeichen im Akku und setzt das Carry-Flag, wenn ein Zeichen gefunden

wurde. Befindet sich an der Cursor-Position dagegen kein Zeichen des CPC-Zeichensatzes, dann enthält der Akku eine Null, das Carry-Flag ist gelöscht.

Des weiteren wird eine Routine benötigt, die uns den Cursor positioniert, damit wir die Zeichen nacheinander lesen können. Diese Funktion wird durch **TXT SET CURSOR, &BB75**, ausgeführt. Wird diese Adresse aufgerufen, so wird der Inhalt des H-Registers als Spalte, der Inhalt von L als Zeile interpretiert. Dabei ist die linke obere Schreibstelle durch &0101 adressierbar.

Allerdings gibt es an dieser Stelle ein kleines Problem. Nachdem wir den Cursor mit unserer Abfrage des Bildschirms einmal über den gesamten Bildschirm gejagt haben, sollte er hinterher schon wieder an seinem ursprünglichen Platz stehen. Dazu müssen wir vor dem ersten Positionieren die Stelle des Cursors ermitteln und merken.
Das kann mittels **TXT GET CURSOR, &BB78**, geschehen. Nach dem Aufruf von TXT GET CURSOR enthält das HL-Registerpaar die derzeitige Cursor-Position. Diesen Wert müssen wir uns merken und zum Abschluß der Hardcopy wieder restaurieren.

Die mittels TXT RD CHAR erhaltenen Zeichen müssen an den Drucker ausgegeben werden. Dazu dient uns **MC SEND PRINTER** mit dem Einsprung bei **&BD31**. Das im Akku befindliche Zeichen wird mit allen benötigten Handshake-Signalen auf den Printerport ausgegeben.

Allerdings erwartet MC SEND PRINTER, daß der Drucker empfangsbereit ist. Ob das der Fall ist, stellt **MC BUSY PRINTER, &BD2E**, für uns fest. Ist der Drucker nicht empfangsbereit, nicht eingeschaltet oder auch nicht angeschlossen, dann kommt MC BUSY PRINTER mit gesetztem Carry-Flag zurück. In diesem Fall muß sie wiederholt aufgerufen werden, bis das Carry gelöscht ist. Dann kann das gewünschte Zeichen ausgegeben werden.

Nun kann aber ja auch der Fall auftreten, daß eine einmal ausgelöste Hardcopy nicht vollständig ausgedruckt werden soll. Dies kann durch Drücken der 'DEL'-Taste unterbrochen werden. Dafür müssen wir aber feststellen können, ob diese Taste gedrückt ist.
Wird **KM TEST KEY, &BB1E,** mit einem gültigen Tastencode im Akku aufgerufen, dann ist nach dem Rücksprung das Zero-Flag gelöscht, wenn die entsprechende Taste gedrückt ist. Andernfalls ist das Zero-Flag gesetzt.

Damit haben wir eigentlich alle System-Routinen, um eine Hardcopy zu schreiben. Aber spätestens beim eifrigen Programmieren wird uns auffallen, das wir ja garnicht wissen, ob 20, 40 oder gar 80 Zeichen zum Zeitpunkt der Hardcopy dargestellt werden.
Gut, man könnte sagen: diese Hardcopy geht nur im Bildschirmmodus x. Das wäre aber eine unschöne Einschränkung.

SCR GET MODE mit dem Einsprung bei **&BC11** teilt uns im Akku und in den beiden Flags Carry und Zero mit, in welchem Darstellungsmodus sich der CPC gerade befindet. Entsprechend können wir eine Hardcopy mit der benötigten Zeichenzahl erstellen.

Doch jetzt zum eigentlichen Programm. Leser ohne Assembler können das am Schluß dieses Kapitels abgedruckte Basicprogramm verwenden. Es enthält beide Hardcopy-Programme in Data-Zeilen.

```
     BB78   GETCRS   EQU   #BB78
     BB75   SETCRS   EQU   #BB75
     BB60   RDCHAR   EQU   #BB60
     BD2E   TSTPTR   EQU   #BD2E
     BD31   PRTCHR   EQU   #BD31
     BC11   GETMOD   EQU   #BC11
     BB1E   TSTKEY   EQU   #BB1E

A100 CD78BB          CALL  GETCRS       ;alte Cursorpos. merken
A103 2264A1          LD    (OLDPOS),HL
```

```
A106  CD11BC          CALL  GETMOD        ;Bildschirmmodus holen
A109  17              RLA                 ;Anzahl der Zeichen/20
A10A  3263A1          LD    (MODE),A      ;und merken
A10D  210101          LD    HL,#0101      ;in die obere linke Ecke
A110  2266A1          LD    (CRSPOS),HL   ;den Cursor
A113  3A63A1  LL1     LD    A,(MODE)
A116  47              LD    B,A           ;1, 2 oder 4 mal
A117  0E14    LOOP    LD    C,20          ;20 Zeichen in eine Zeile
A119  C5      LLOOP   PUSH  BC
A11A  E5              PUSH  HL
A11B  CD75BB          CALL  SETCRS        ;Cursor platzieren
A11E  E1              POP   HL
A11F  CD60BB          CALL  RDCHAR        ;und das Zeichen
A122  C1              POP   BC            ;bestimmen
A123  3802            JR    C,GOOD        ;gültiges Zeichen?
A125  3E20            LD    A,32          ;sonst Leerzeichen
A127  CD58A1          GOOD  CALL  PRTOUT  ;ausgeben
A12A  E5              PUSH  HL
A12B  C5              PUSH  BC
A12C  3E42            LD    A,66          ;ESC gedrückt?
A12E  CD1EBB          CALL  TSTKEY
A131  C1              POP   BC
A132  E1              POP   HL
A133  201C            JR    NZ,EXIT       ;wenn ja, dann Ende
A135  24      WEITER  INC   H
A136  0D              DEC   C
A137  20E0            JR    NZ,LLOOP      ;20 Zeichen gedruckt?
A139  10DC            DJNZ  LOOP          ;ganze Zeile?
A13B  3E0D            LD    A,#0D         ;CR/LF ausgeben
A13D  CD58A1          CALL  PRTOUT
A140  3E0A            LD    A,#0A
A142  CD58A1          CALL  PRTOUT
A145  2A66A1          LD    HL,(CRSPOS)   ;Cursorpos. für
A148  2C              INC   L             ;nächste Reihe
A149  2266A1          LD    (CRSPOS),HL   ;bestimmen
A14C  7D              LD    A,L
A14D  FE1A            CP    26            ;25 Zeilen gedruckt?
A14F  20C2            JR    NZ,LL1
A151  2A64A1  EXIT    LD    HL,(OLDPOS)   ;wenn ja, dann alte
```

```
A154  CD75BB           CALL  SETCRS     ;Cursorpos. restaurieren
A157  C9               RET              ;und zurück
A158  C5      PRTOUT   PUSH  BC
A159  CD2EBD  P1       CALL  TSTPTR     ;Printer Busy?
A15C  38FB             JR    C,P1
A15E  CD31BD           CALL  PRTCHR     ;Zeichen ausgeben
A161  C1               POP   BC
A162  C9               RET
A163  00      MODE     DEFB  0
A164  0000    OLDPOS   DEFW  0000
A166  0000    CRSPOS   DEFW  0000
```

Durch die Kommentare im Listing sollte das Programm einfach zu verstehen sein. Die einzige Besonderheit stellt die Methode zur Berechnung der Anzahl pro Zeile auszugebener Zeichen dar. Darum wollen wir noch kurz darauf eingehen.

Nach dem Aufruf von SCR GET MODE enthält der Akku je nach Modus 0, 1 oder 2. Zusätzlich haben Carry- und Zero-Flag folgende Zustände:

```
Mode 0 = Carry 1, Zero 0
Mode 1 = Carry 0, Zero 1
Mode 2 = Carry 0, Zero 0
```

Durch den Befehl SLA wird der Inhalt des Akkus um ein Bit nach links verschoben. Das entspricht einer Multiplikation mit zwei. Zusätzlich wird der Zustand des Carry-Flags in das freigewordene Bit 0 des Akku übertragen, das 'herausgefallene' Bit 7 wird in das Carry übernommen.
Im Mode 0 wird die im Akku befindliche 0 rotiert. Das hat auf den Inhalt des Akkus keinen Einfluß. Da aber das von SCR GET MODE gesetzte Carry-Flag in das Bit 0 des Akkus übertragen wird, enthält der Akku nach diesem Befehl eine 1. Diese 1 bewirkt, daß ein mal 20 Zeichen in einer Zeile ausgedruckt werden.
Im Mode 1 enthält der Akku eine 1, das Carry ist in diesem Mode gelöscht. Nach dem SLA enthält der Akku eine 2. Somit werden 2 mal 20 Zeichen in einer Zeile ausgegeben. Analog

sind die Verhältnisse im Mode 2. Allerdings ist das Ergebnis des SLA eine 4 im Akku, welche 4 mal 20 Zeichen in einer Druckzeile bewirken.

Etwas anders sind die Verhältnisse, wenn es darum geht, eine Grafik-Hardcopy zu erzeugen. Hier können uns die Routinen TXT SET CURSOR und TXT RD CHAR nicht weiterhelfen.
Als erste Aktion wird mit **GRA INITIALISE** der Grafik-Modus eingeschaltet. Danach bestimmen wir mit **GRA GET PAPER** die Farbnummer des Hintergrundes. Mit diesem Wert werden alle Punkte des Bildschirms verglichen. Ist die Farbe eines Pixels vom Hintergrund verschieden, dann wird auf dem Papier ein Punkt erzeugt.

Leider verfügt der CPC nur über einen 7-Bit-Druckeranschluß. Dadurch ergeben sich gewisse Komplikationen.
Zunächst bedeutet es, daß wir mit einem Male 7 Punkte, die untereinander angeordnet sind, auf den Drucker ausgeben können. Insgesamt hat die Grafik des CPC eine vertikale Auflösung von 200 Punkten. Das läßt sich aber nicht ohne Rest durch 7 teilen. Wir erhalten einen Rest, also verbleibende Pixelreihen, die gesondert behandelt werden müssen. Dafür gibt es aber keine Probleme mit unterschiedlichen Textmodi.
Ein weiteres Problem mit dem 7-Bit-Ausgang ergibt sich bei der Befehlsübermittlung an den Drucker. Das Einschalten der Grafik mit ESC L erfordert für die vorhandenen 640 Pixel pro Zeile eine Angabe, die mit dem CPC garnicht zu übertragen ist. Um die geforderte Anzahl der Grafikpunkte zu erhalten, lautet die Steuersequenz für den Drucker:

```
PRINT #8,CHR$(27);"L";CHR$(128)CHR$(2)
```

Der Wert 128 stellt das Problem dar. Binär ausgedrückt ist 128 eine Zahl mit gesetztem achtem Bit. Alle anderen Bits sind Null. Würden wir diesen Wert an den Drucker schicken, so erhielte dieser nur eine Null, da das achte Bit ja als Srtobe verwendet wird und nicht an den Drucker ausgegeben wird.

Wir haben dies Problem in der nicht ganz eleganten Weise umgangen, indem wir nur 639 Punkte, in horizontaler Richtung ausgeben. Das ist zwar ein Punkt weniger, als auf dem Bildschirm vorhanden ist, dadurch reduziert sich aber der erste zu übertragende Wert auf 127.

Bevor wir nun zum Listing der Grafik-Hardcopy kommen, müssen wir noch auf eine Besonderheit hinweisen.
Obwohl der Bildschirm physikalisch nur 200 Rasterzeilen darstellt wird im CPC bei allen Grafik-Routinen mit einer vertikalen Auflösung von 400 Punkten gerechnet. Dadurch ergibt sich ein deutlich besseres Verhältnis von X- zu Y-Richtung, als wenn nur mit den tatsächlich vorhandenen 200 Zeilen gerechnet würde.
Der Effekt ist gut sichtbar, wenn Sie einmal das im Basic-Handbuch zum CPC abgedruckte Programm zum Zeichen eines Kreises ausprobieren. Der Kreis ist tatsächlich (fast) rund. Ohne diese Korrektur würde eine auf der Seite liegende Ellypse erzeugt.
Diese Korrektur muß auch in unserer Hardcopy stattfinden, allerdings in der genau entgegen gesetzten Form. Auch wir ermitteln die Grafik-Koordinaten im Raster 400x640 Punkte, auf dem Drucker werden aber nur 200 Punkte in vertikaler Richtung ausgegeben, um die Verzerrungen gering zu halten.

```
        A000            ORG     #A000
        BBBA    GRINIT  EQU     #BBBA
        BBE7    GETPAP  EQU     #BBE7
        BBF0    TSTPOI  EQU     #BBF0
        BD2B    PRINTO  EQU     #BD2B
        BD2E    TSTPTR  EQU     #BD2E
        BB1E    TSTKEY  EQU     #BB1E

A000    CDBABB          CALL    GRINIT          ;Grafik-Modus einschalten
A003    CDE7BB          CALL    GETPAPER        ;Hintergrundfarbe bestimmen
A006    32BDA0          LD      (PAPER),A
A009    CD6CA0          CALL    INITP           ;Drucker auf 7/72 Zoll einstellen
A00C    218F01          LD      HL,399          ;wir fangen oben und
A00F    22BEA0          LD      (Y-MERK),HL
```

```
A012  110000          LD    DE,0          ;links an zu drucken
A015  3E07            LD    A,7           ;leider nur mit 7 Nadeln
A017  32C0A0          LD    (ANZAHL),A
A01A  CD7CA0  LLOOP   CALL  PRTESC        ;ESC-Seq. für Grafik
A01D  0E00    LL1     LD    C,0           ;C enthält Bitmuster für Printer
A01F  3AC0A0          LD    A,(ANZAHL)
A022  47              LD    B,A           ;B = Dotreihen-Zähler
A023  E5      BYTLP   PUSH  HL
A024  D5              PUSH  DE
A025  C5              PUSH  BC
A026  CDF0BB          CALL  TSTPOINT      ;Farbe des Pixels an der
                                          ;Position (hl/de) bestimmen
A029  C1              POP   BC
A02A  D1              POP   DE
A02B  21BDA0          LD    HL,PAPER
A02E  BE              CP    (HL)          ;Pixelfarbe=Hintergrundfarbe?
A02F  E1              POP   HL
A030  37              SCF                 ;wenn Pixel <> Paper, dann
A031  2001            JR    NZ,DOT        ;Carry-Flag setzen, sonst
A033  A7              AND   A             ;Carry loeschen
A034  CB11    DOT     RL    C             ;Carry ins unterste Bit
A036  2B              DEC   HL            ;des C-Registers schieben,
A037  2B              DEC   HL            ;HL=HL-2, nächster Punkt,
A038  10E9            DJNZ  BYTLP         ;und das ganze 7 Mal
A03A  CDAFA0          CALL  TEST          ;Sonderbehandl. der letzten
A03D  79              LD    A,C           ;Bitmuster in Akku übertragen
A03E  CDA6A0          CALL  PRINT         ;und drucken
A041  13              INC   DE
A042  E5              PUSH  HL
A043  217F02          LD    HL,639        ;eine Zeile gedruckt?
A046  37              SCF
A047  ED52            SBC   HL,DE
A049  E1              POP   HL
A04A  3805            JR    C,NXTROW
A04C  2ABEA0          LD    HL,(Y-MERK)
A04F  18CC            JR    LL1
A051  23      NXTROW  INC   HL            ;Sonderbehandl. der
A052  7C              LD    A,H           ;letzten 4
A053  B5              OR    L
```

```
A054  C8                   RET   Z
A055  2B                   DEC   HL
A056  110000               LD    DE,0          ;Vorbereiten der nächsten
A059  22BEA0               LD    (Y-MERK),HL   ;Druckzeile
A05C  3E07                 LD    A,7
A05E  BD                   CP    L             ;letzte 7-er Reihe?
A05F  20B9                 JR    NZ,LLOOP
A061  7C                   LD    A,H
A062  B4                   OR    H
A063  20B5                 JR    NZ,LLOOP
A065  3E04                 LD    A,4           ;dann nur noch 4Reihen
A067  32C0A0               LD    (ANZAHL),A
A06A  18AE                 JR    LLOOP
A06C  3E1B       INITP     LD    A,27          ;für NLQ/MX/RX/FX
A06E  CDA6A0               CALL  PRINT         ;ESC A 7, um den richtigen
A071  3E41                 LD    A,65          ;Zeilenvorschub zu bekommen
A073  CDA6A0               CALL  PRINT
A076  3E07                 LD    A,7
A078  CDA6A0               CALL  PRINT
A07B  C9                   RET
A07C  E5         PRTESC    PUSH  HL            ;DEL-Taste gedrückt?
A07D  3E42                 LD    A,66          ;wenn ja, dann HC abbrechen
A07F  CD1EBB               CALL  TSTKEY
A082  E1                   POP   HL
A083  2802                 JR    Z,NOKEY       ;DEL war nicht gedrückt
A085  E1                   POP   HL            ;Stack manipulieren
A086  C9                   RET                 ;um an den Ret zu kommen
A087  3E0D       NOKEY     LD    A,#0D         ;CR/LF ausgeben
A089  CDA6A0               CALL  PRINT
A08C  3E0A                 LD    A,10
A08E  CDA6A0               CALL  PRINT
A091  3E1B                 LD    A,27          ;ESC L 127 2 = Grafik
A093  CDA6A0               CALL  PRINT         ;mit 639 Punkten
A096  3E4C                 LD    A,76
A098  CDA6A0               CALL  PRINT
A09B  3E7F                 LD    A,127
A09D  CDA6A0               CALL  PRINT
A0A0  3E02                 LD    A,2
A0A2  CDA6A0               CALL  PRINT
```

```
A0A5  C9                RET
A0A6  CD2EBD  PRINT     CALL  TSTPTR       ;Drucker Busy?
A0A9  38FB              JR    C,PRINT
A0AB  CD2BBD            CALL  PRINTOUT     ;Zeichen drucken
A0AE  C9                RET
A0AF  3AC0A0  TEST      LD    A,(ANZAHL)   ;Behandlung der letzten
A0B2  FE07              CP    7            ;vier Dotreihen
A0B4  C8                RET   Z
A0B5  AF                XOR   A
A0B6  CB11              RL    C            ;dreimal 0 über Carry
A0B8  CB11              RL    C            ;ins C-Reg. schieben
A0BA  CB11              RL    C
A0BC  C9                RET
A0BD  00      PAPER     DEFB  0
A0BE  0000    Y-MERK    DEFW  0000
A0C0  00      ANZAHL    DEFB  0
```

Zum Abschluß jetzt der versprochene Basic-Lader. Damit können Sie die Programm auch einsetzen, wenn Sie keinen Monitor oder Assembler besitzen.

```
100 REM Grafik-Hardcopy fuer cpc mit NLQ/MX/RX/FX
110 REM hardcopy wird mit 'CALL &A000' aufgerufen
120 REM text-hardcopy fuer den cpc
130 REM hardcopy wird mit 'call &a100' aufgerufen
140 FOR i=&A000 TO &A0BF
150 READ byte:POKE i,byte:s=s+byte:NEXT
160 DATA &cd,&ba,&bb,&cd,&e7,&bb,&32,&bd
165 DATA &a0,&cd,&6c,&a0,&21,&8f,&01,&22
170 DATA &be,&a0,&11,&00,&00,&3e,&07,&32
175 DATA &c0,&a0,&cd,&7c,&a0,&0e,&00,&3a
180 DATA &c0,&a0,&47,&e5,&d5,&c5,&cd,&f0
185 DATA &bb,&c1,&d1,&21,&bd,&a0,&be,&e1
190 DATA &37,&20,&01,&a7,&cb,&11,&2b,&2b
195 DATA &10,&e9,&cd,&af,&a0,&79,&cd,&a6
200 DATA &a0,&13,&e5,&21,&7f,&02,&37,&ed
205 DATA &52,&e1,&38,&05,&2a,&be,&a0,&18
210 DATA &cc,&23,&7c,&b5,&c8,&2b,&11,&00
215 DATA &00,&22,&be,&a0,&3e,&07,&bd,&20
220 DATA &b9,&7c,&b4,&20,&b5,&3e,&04,&32
```

```
225 DATA &c0,&a0,&18,&ae,&3e,&1b,&cd,&a6
230 DATA &a0,&3e,&31,&cd,&a6,&a0,&00,&00
235 DATA &00,&00,&00,&c9,&e5,&3e,&42,&cd
240 DATA &1e,&bb,&e1,&28,&02,&e1,&c9,&3e
245 DATA &0d,&cd,&a6,&a0,&3e,&0a,&cd,&a6
250 DATA &a0,&3e,&1b,&cd,&a6,&a0,&3e,&4c
255 DATA &cd,&a6,&a0,&3e,&7f,&cd,&a6,&a0
260 DATA &3e,&02,&cd,&a6,&a0,&c9,&cd,&2e
265 DATA &bd,&38,&fb,&cd,&2b,&bd,&c9,&3a
270 DATA &c0,&a0,&fe,&07,&c8,&af,&cb,&11
275 DATA &cb,&11,&cb,&11,&c9,&00,&00,&00
280 IF s<>23151 THEN PRINT "error in grafik-hc":END
290 PRINT "grafik-hc korrekt geladen"
300 s=0:FOR i=&A100 TO &A162
310 READ byte:POKE i,byte:s=s+byte:NEXT
320 DATA &cd,&78,&bb,&22,&64,&a1,&cd,&11
325 DATA &bc,&17,&32,&63,&a1,&21,&01,&01
330 DATA &22,&66,&a1,&3a,&63,&a1,&47,&0e
335 DATA &14,&c5,&e5,&cd,&75,&bb,&e1,&cd
340 DATA &60,&bb,&c1,&38,&02,&3e,&20,&cd
345 DATA &58,&a1,&e5,&c5,&3e,&42,&cd,&1e
350 DATA &bb,&c1,&e1,&20,&1c,&24,&0d,&20
355 DATA &e0,&10,&dc,&3e,&0d,&cd,&58,&a1
360 DATA &3e,&0a,&cd,&58,&a1,&2a,&66,&a1
365 DATA &2c,&22,&66,&a1,&7d,&fe,&1a,&20
370 DATA &c2,&2a,&64,&a1,&cd,&75,&bb,&c9
375 DATA &c5,&cd,&2e,&bd,&38,&fb,&cd,&31
380 DATA &bd,&c1,&c9
390 IF s<>11873 THEN PRINT "error in text-hc":END
400 PRINT "text-hc korrekt geladen"
```

2.4 Interrupts im Betriebssystem

Die wohl schnellste und leistungsfähigste Möglichkeit, innerhalb eines Betriebssystems auf bestimmte Ereignisse zu reagieren, ist die Interrupt-Technik.
Sie wissen sicher, was das ist. Falls nicht, hier das Wesentliche in dürren Worten:
Ein Interrupt (auf Deutsch: eine Unterbrechung) ist in der Regel ein Hardwareereignis, welches ein laufendes Programm über sein Auftreten informiert. Abhängig von diesem Ereignis soll die Software jetzt zugeordnete Aktionen durchführen, und zwar, je nach Dringlichkeit, möglichst schnell. Eine derartige Aktivität ist z.B. das Scrollen des Bildschirms während der Dunkelphase des Elektronenstrahls, damit es für den Betrachter möglichst flimmerfrei über die Bühne geht.
Diese Interupttechnik bietet den Vorteil, den übrigen Programmablauf wirklich nur für eine notwendige Aktion zu unterbrechen, so daß die Software nicht dauernd nachschauen muß, ob nun etwas passiert ist oder nicht.
Es gibt naturgemäß viele Möglichkeiten, eine solche Fähigkeit in ein Betriebssystem zu integrieren (wie böse Zungen behaupten, ebenso viele, wie es Programmierer gibt), aber wir müssen zugeben, daß uns eine solche Variante, wie sie uns im CPC geboten wird, noch nicht untergekommen ist.

Es handelt sich hier um einen raffinierten Mix von Hardwareinterrupt (Unterbrechung, wenn nötig) und Polling (regelmäßig nachschauen, was los ist).
Wie dringend eine 'Anfrage' behandelt werden soll, entscheidet der Programmierer der zugehörigen Routine. Im Klartext:
Es gibt in der Maschine nur einen einzigen Interrupt, und das ist der Timer (im System 'Fast Ticker' genannt), der alle 1/300s eine Unterbrechung erzeugt. Alles Weitere ergibt sich hieraus, wie Sie sehen werden.

Es ist hier an der Zeit, einige neue Begriffe einzuführen, auf die Sie ab hier und auch im Rom-Listing öfter stoßen werden.

1. **EVENT** bedeutet ganz einfach **Ereignis**. Verstehen Sie in diesem Zusammenhang bitte eine Art softwaregesteuerten Interrupt.
2. **FRAME FLYBACK** ist nichts anderes als der oben erwähnte **Strahlrücklauf des Bildschirmes**, was alle 1/50s geschieht.
3. **TICKER** ist ein **Vielfaches des Fast Ticker** und erscheint ebenfalls alle 1/50s.

Das Ganze wird einfach so gehandhabt, daß der Programmierer, also u.U. Sie, bestimmt, welche Routinen seines Programmes wie oft zum Zeitpunkt Frame Flyback, Ticker oder gar Fast Ticker angesprungen werden sollen, und zwar automatisch, also ohne sein weiteres Zutun. Als Vorbereitung dazu ist dem Betriebssystem lediglich, neben ein paar weiteren Kleinigkeiten, einmal die Adresse der Routine(n) mitzuteilen. Das Weitere findet sich.
Diese bereitzustellende Information nennt sich **EVENT BLOCK**. Hierin ist hinterlegt, wie oft und wann die Routine aufgerufen werden soll, ob vor evtl. weiteren Routinen (Prioritätensteuerung) oder ob es damit Zeit hat, usw.

Bei jedem Eintritt von Ticker, Fast Ticker oder Frame Fly schaut das Betriebssystem nach, ob es zugehörige Event-Blocks gibt. Falls ja, werden sie, entsprechend ihrer Priorität, aufgerufen. Einige Event-Blocks gibt es übrigens immer, z.B. die Aktion, zum Zeitpunkt Frame Fly das Farbregister zu versorgen.

Die einem bestimmten Ereignis zugeordneten Blocks sind durch Pointer miteinander verkettet, so daß sich das Betriebssystem von einem zum anderen durchhangeln kann. Demzufolge ist es ohne Bedeutung, an welcher Adresse ein solcher Block steht, solange er sich nur **in den zentralen 32k des Ram** befindet. Diese kleine Einschränkung muß deswegen gemacht werden, da dieser Bereich der einzige ist, auf den jederzeit, unabhängig von der übrigen Rom-Konfiguration, zugegriffen werden kann.
Soll ein solcher Block ausgeführt werden, so wird er in eine andere Kette eingereiht, die sog. **Pending Queue**. Dieser Vorgang heißt **Kicken**.

Die Pending Queue wird am Ende der systemeigenen Interrupt-Routine abgearbeitet. Sie sagen sich sicher, daß ein vorhandener Block selbstverständlich ausgeführt werden soll, wozu also noch extra in eine besondere Schlange einreihen?
Nun, so selbstverständlich ist das nicht, denn Sie haben durchaus die Möglichkeit, die Behandlung eines Blockes für eine Weile auszusetzen, ohne daß Sie Ihn aus der Primärqueue ausketten müssen, was übrigens bei Event-Blocks der Ticker-Queue besonders komfortabel zu machen ist.
Übrigens: Nicht daß Sie glauben, es gäbe in dem Rechner nur diesen Timer-Interrupt. Hardware-Freaks haben ohne weiteres die Möglichkeit, via Expansion-Bus einen Interrupt zu erzeugen (asynchron), nur sollte dann auch eine entsprechende Routine vorhanden sein, die den zugehörigen Event-Block 'kickt'.

Werden wir doch einmal konkret. Was ist zu tun, wenn von diesem Mechanismus Gebrauch gemacht werden soll?
Zunächst wird natürlich ein Event-Block angelegt, dessen Aufbau wie folgt vorgeschrieben ist. Allen Event-Arten ist folgender Teil gemeinsam:

```
Byte 0+1 Kettungsadresse für die Pending Queue. Dieses  Feld
         darf nur vom Betriebssystem versorgt werden!
Byte 2   Zähler
         Solange der Zähler > 0 ist, verbleibt der Block  in
         der Pending Queue,  d.h.  die  Routine  wird  sooft
         ausgeführt, bis er =0 ist.
         Ist der Zähler < 0 (d.h. > 127), bleibt  der  Block
         in der betreffenden Kette (Ticker usw.).  Auch  ein
         Kicken führt in diesem Fall  nicht  zur  Ausführung
         der Routine, was ansonsten ein Erhöhen des  Zählers
         und damit  auch  den  Ansprung  bei  nächster
         Gelegenheit zur Folge hätte.
Byte 3   Klasse
         Bit0 = 1 = Bei der Sprungadresse handelt es sich um
         eine Near Address, d.h. sie liegt im zentralen Ram,
```

	bzw. unteren Rom.
	Bit0 = 0 = Die Sprungadresse ist eine Far Address, also im oberen Rom zu suchen.
	Die Bits 1-4 bestimmen die Priorität.
	Bit5 muß immer =0 sein!
	Bit6 = 1 = Express. Die Express-Events haben eine höhere Priorität als normale Events mit der höchsten Priorität.
	Bit7 = 1 = Asynchroner Event. Diese Events haben keine Warteschlange, sondern werden beim Kicken (KL EVENT) sofort in die Interrupt Pending Queue eingereiht. Handelt es sich sogar um einen Express, wird er auf der Stelle ausgeführt, ansonsten erst am Ende der Interrupt-Routine.
	Achtung: Die Routine für asynchrone Express-Events muß im zentralen Ram liegen!
Byte 4+5	Adresse der Routine
Byte 6	Rom Select, wenn Sprungadresse vom Typ Far ist, sonst unbenutzt.
Byte 7	Hier beginnt das Benutzerfeld, welches beliebig lang sein darf. Es kann zur Übergabe von Parametern an die Routine dienen. Beim Ansprung einer Event-Routine enthält hl die Adresse von Byte 5 des Event-Blocks, wenn es sich um eine Near Address handelte, ansonsten die Adresse von Byte 6.
	Dieser Umstand ermöglicht es, mehrere Blocks für die gleiche Routine anzulegen, welche anhand der Parameter sehen kann, von welchem Block sie gerufen wurde.

Abhängig vom Typ des Events, also Ticker, Fast Ticker oder Frame Fly, werden noch zwei oder sechs Bytes dem gemeinsamen Teil vorangestellt. Im Falle Fast Ticker und Frame Fly sind es nur zwei Bytes für die Kettung (nicht verändern!) in der Fast Ticker List, bzw. Frame Fly List.
Die sechs Bytes für den Ticker haben folgende Bedeutung:

Byte 0+1	Kettung für Ticker List (nicht verändern!)
Byte 2+3	Tick Count bestimmt, wie oft ein Ticker erscheinen

```
        muß, bevor der Block einmal gekickt wird.
Byte 4+5 Reload Count gibt an, mit  welchem  Wert  der  Tick
        Count nach Ablauf wieder geladen werden soll.
```

Nachdem Sie also Ihren Block mit den Werten versorgt haben, soweit sie Ihnen bekannt sind (das sollten wenigstens die letzten 5 Bytes (Event Count=0) des gemeinsamen Teils sein), brauchen Sie nur noch hl mit der Startadresse Ihres Blockes laden (im Falle Ticker gehört noch der Tick Count nach de und der Reload Count nach bc) und, je nach dem, die Routine **KL ADD TICKER, KL ADD FAST TICKER** oder **KL ADD FRAME FLY** anspringen, und ab geht die Post.
Zum Aushängen des Blocks aus der Liste benutzen Sie die Routinen **KL DEL TICKER** usw., wobei hl wieder die Adresse, diesmal die des zu entfernenden Blocks, enthalten muß.

Versuchen Sie es einmal und schauen Sie nach, wie das Betriebssystem es macht, denn immer wiederkehrende Prozesse werden auch dort über den Event-Mechanismus abgehandelt.

2.5 Das Betriebssystem-ROM

Sicher haben Sie schon bei der Aufstellung der Betriebssystem-Vektoren (Kapitel 2.1) wie des Betriebssystem-RAMs (Kapitel 2.2) bemerkt, daß die Unterschiede zwischen dem CPC 664 und CPC 6128 minimal sind. Deshalb haben wir uns entschieden - um keine Seitenschinderei zu betreiben - bei der Betrachtung des Betriebssystems unser besonderes Augenmerk auf den CPC 6128 zu richten. Das bedeutet nicht, daß alle CPC 664-Besitzer jetzt das Buch schließen und in den Schrank stellen können, sondern lediglich, daß diese ein wenig aufpassen müssen, ob die abgedruckten Kommentare unmittelbar zu Ihrem ROM-Listing passen oder ob sich die Adressen um ein paar Bytes verschoben haben. Da Sie ja soweit in der Maschinensprache zuhause sind, daß Sie es verstehen mit dem Betriebssystem Ihre Experimente anzustellen, dürfte Ihnen diese Transferleistung nicht schwerfallen. Sind Sie aber noch nicht so sattelfest, sollten Sie ohnehin ausschließlich die Vektoren verwenden.

Sie finden im folgenden die Kommentare zum Betriebssystem des CPC 6128. Diese Kommentare sind, betrachtet man sie isoliert, nicht sehr aussagekräftig; erzeugt man sich aber mit dem im Anhang dieses Buches abgedruckten DISASSEMBLER ein Listing von dem zu untersuchenden ROM-Bereich und bringt dann die Adressen des Listings mit denen der Kommentare zur Deckung, so fügt sich beides zu einem sinnvollen Ganzen zusammen.

Diese Art der Kommentare zu Betriebssystemen wird mit steigender Komplexität dieser Gebilde sicher bald gang und gäbe sein, will man das gesamte Betriebssystem beschreiben und nicht nur Teilbereiche beleuchten.

2.5.1 Kernel (KL)

Das Kernel hält, wie der Name schon vermuten läßt, die Fäden in der Hand. Es ist für die Ablaufsteuerung zuständig, das will heißen, für die Interruptbehandlung und der damit verbundenen Events. Ebenso fällt ihm die Aufgabe zu, Restarts zu bearbeiten, ROM-Erweiterungen einzuhängen und zwischen Speicherkonfigurationen umzuschalten.

Besonders die Routinen, die im Zusammenhang mit dem Event-Mechanismus stehen, dürfen für den Anwender interessant sein.

0000 ********************************** RST 0 RESET ENTRY

Nach dem Einschalten des Systems beginnt der Prozessor hier mit der Abarbeitung des Programms. Ein Aufruf von RST 0 bewirkt ein vollständiges Zurücksetzen des Systems.

0000 U ROM disable, Mode 1, reset Teiler
0005 Reset Cont'd

0008 ************************************** RST 1 LOW JUMP

Dient zum Aufruf einer Routine im Betriebssystem oder im darunterliegenden RAM. Direkt hinter dem RST-Befehl muß die Adresse der aufzurufenden Routine stehen. Da für den Bereich von &0000 bis &3FFF 14 Adreßbits ausreichen, werden die Bits 14 und 15 zur Auswahl von ROM oder RAM benutzt. Durch das gesetzte Bit 15 ist im Adreßbereich &C000 bis &FFFF RAM selektiert, wohingegen durch das gelöschte Bit 14 das Betriebssystem ausgewählt wird.

0008 (0430) RST 1 LOW JUMP CONT'D
000B (042A) KL LOW PCHL CONT'D

000E Rücksprungadresse manipulieren
000F Entspricht jp (bc)

0010 *************************************** RST 2 SIDE CALL

Dient zum Aufruf einer Routine in einem Expansion-ROM. RST 2 wird benutzt, wenn ein Programm, das als ROM-Erweiterung angeschlossen ist, mehr als 16 K benötigt.

0010 (04C3) RST 2 LOW SIDE CALL CONT'D
0013 (04BD) KL SIDE PCHL CONT'D
0016 Rücksprungadresse manipulieren
0017 Entspricht jp (de)

0018 *************************************** RST 3 FAR CALL

Es kann eine Routine irgendwo im ROM oder RAM aufgerufen werden. Dazu muß hinter dem RST 3-Befehl die Zwei-Byte-Adresse eines Parameterblocks (drei Byte lang) folgen. Die ersten beiden Bytes des Parameterblocks enthalten die Adresse der aufzurufenden Routine. Das dritte Byte gib den ROM/RAM-Status an.

0018 (046D) RST 3 LOW FAR CALL CONT'D
001B (045F) KL FAR PCHL CONT'D

0020 *************************************** RST 4 RAM LAM

Mit dem RST 4 können Sie von einem Maschinenprogramm aus den Inhalt des RAMs lesen, dabei spielt der jeweils gewählte ROM-Zustand keine Rolle. Der RST 4-Befehl ersetzt quasi LD A,(HL), dazu muß hl die Adresse der zu lesenden Speicherstelle enthalten.

0020 (056C) RST 4 RAM LAM CONT'D
0023 (0467) KL FAR ICALL CONT'D

0028 ************************************ RST 5 FIRM JUMP

Ermöglicht das Springen zu einer Routine im Betriebssystem. Die zugehörige Einsprungadresse muß dem RST 5-Befehl unmittelbar folgen. Bevor der Sprung auf die gewünschte Routine erfolgt, wird das Betriebssystem-ROM selektiert und nach dem Verlassen der Routine wieder abgeschaltet.

0028 (04DB) RST 5 FIRM JUMP CONT'D

0030 ********************************* RST 6 USER RESTART

Die Bytes &0030 bis &0037 stehen dem Benutzer zur Verfügung. Beim Einschalten des Systems ist ein RST 0 voreingestellt.

0030 RST 0 nach High Kernel Restore

0038 **************************** RST 7 INTERRUPT ENTRY

Einsprung für Hardware-Interrupts.

0038 (03E7) RST 7 INTERRUPT ENTRY CONT'D
003B EXT INTERRUPT

0040 *************************** bis hier wird ins RAM kopiert

0040 L ROM disable

0044 ****************************** Restore High Kernel Jumps

0044 003F
0047 bis
0048 0000
0049 ins RAM
004A kopieren
004C RST 0 nach
004E 0030

0051 Jump
0054 Block
0057 kopieren

005C ************************************** KL CHOKE OFF

Kernel zurücksetzen, Event-Warteschlangen löschen und einiges mehr.

005D (lfd. ROM-Konfiguration)
0060 (Einsprung lfd. ROM)
0064 Firmware-
0066 RAM
0069 bis
006B B8CD
006C löschen
0071 war ein ROM on?
0072 ja springe
007C falls hl=0
007D Default laden
0080 (lfd. Exp.-ROM)
0083 (lfd. ROM-Konfiguration)
0086 (Einsprung lfd. ROM)
0089 Params für
008C RST 3 laden
0095 FAR CALL
0096 dw B8D7

0099 ************************************** KL TIME PLEASE

Wieviel Zeit ist abgelaufen?

009A (Timer high)
009E (Timer low)

00A3 ** KL TIME SET

Setzen der Zeit auf einen vorgegebenen Wert.

00A4 lade Akku mit 0 und setze Flags zurück
00A5 (Timerflag)
00A8 (Timer high)
00AC (Timer low)

00B1 ** Scan Events

00B1 Timer low
00B4 update
00B5 Timer
00BA Port B
00BC VSYNC?
00BD nein springe
00BF (Start Frame Fly Chain)
00C2 Highbyte nach Akku
00C3 ist Akku 0?
00C4 Akku nicht 0 springe zu Kick Event
00C7 (Start Fast Ticker Chain)
00CA Highbyte nach Akku
00CB ist Akku 0?
00CC Akku nicht 0 springe zu Kick Event
00CF Scan Sound Queues
00D2 Count for Ticker
00D9 Update Key State Map
00DC (Start Ticker Chain)
00DF Highbyte nach Akku
00E0 ist Akku 0?
00E1 Akku 0 springe
00E2 diverse Flage für Int. Routine
00E5 Ticker Chain muß noch
00E7 bearbeitet werden
00F2 (Start Int Pending Queue)
00F8 diverse Flags für Int. Routine
010A (sp save)
010E Timer low

0114 diverse Flags für Int. Routine
011D (Start Int Pending Queue)
0120 Highbyte nach Akku
0121 ist Akku 0?
0122 Akku 0 springe
0127 (Start Int Pending Queue)
0132 diverse Flags für Int Routine
0135 Ticker Queue pending ?
0137 nein springe
013D Ticker Chain bearbeiten
0142 diverse Flags für Int Routine
0145 Highbyte nach Akku
0146 noch etwas bearbeiten?
0147 ja springe
0149 Flags löschen
014E sp rückladen

0153 ** Kick Event

0158 KL EVENT
015D KL EVENT
0161 Kick Event

0163 *********************************** KL NEW FRAME FLY

Eventblock einrichten und einhängen.

0166 KL INIT EVENT

016A ********************************* KL ADD FRAME FLY

Eventblock einhängen.

016A Start Frame Fly Chain
016D Add Event

0170 *********************************** KL DEL FRAME FLY

Eventblock aushängen.

0170 Start Frame Fly Chain
0173 Delete Event

0176 ********************************* KL NEW FAST TICKER

Eventblock einrichten und einhängen (vgl. KL NEW FRAME FLY).

0179 KL INIT EVENT

017D ******************************** KL ADD FAST TICKER

Eventblock einhängen (vgl. KL ADD FRAME FLY).

017D Start Fast Ticker Chain
0180 Add Event

0183 ********************************* KL DEL FAST TICKER

Eventblock aushängen (vgl. KL DEL FRAME FLY).

0183 Start Fast Ticker Chain
0186 Delete Event

0189 ********************************** Ticker Chain bearbeiten

0189 (Start Ticker Chain)
018C Highbyte nach Akku
018D ist Akku 0?
018E Akku 0 springe
01A4 KL EVENT

01B3 ************************************** KL ADD TICKER

Tickerblock einhängen.

01BF Start Ticker Chain
01C2 Add Event

01C5 ************************************** KL DEL TICKER

Tickerblock aushängen.

01C5 Start Ticker Chain
01C8 Delete Event

01D2 ************************************** KL INIT EVENT

Eventblock einrichten.

01E2 ** KL EVENT

Eventblock 'kicken'.

01E7 Event Cnt >127/<0
01EB Event Cnt >0&<127
01F1 Sync Event einhängen

0219 ** KL DO SYNC

Eventroutine ausführen.

021F (0467) KL FAR INCALL CONT'D

0227 ************************************** KL SYNC RESET

Sync Pending Queue löschen.

022E ************************************ Sync Event einhängen

022F	Priorität nach b
0230	auszuführender Befehl
0236	Adresse des nächsten
0237	Event Blocks
0238	nach de bringen
0240	lfd. Priorität > gefundene
0241	Priorität?
0242	nein springe

0255 ** KL NEXT SYNC

Der nächste bitte.

0256	(Start Sync Pending Queue)
0259	Highbyte nach Akku
025A	ist Akku 0?
025B	Akku 0 springe
0263	(Priorität lfd. Event)
026B	(Priorität lfd. Event)
026E	(Start Sync Pending Queue)

0276 ** KL DONE SYNC

Eventroutine fertig.

0276	(Priorität lfd. Event)
027E	Sync Event einhängen

0284 ******************************** KL DEL SYNCHRONOUS

Bestimmten Block aus der Pending Queue löschen.

0284	KL DISARM EVENT
0287	Start Sync Pending Queue
028A	Delete Event

028D *********************************** KL DISARM EVENT

Eventblock verriegeln (Zähler negeativ).

0294 ************************************ KL EVENT DISABLE

Sperren der normalen gleichzeitigen Ereignisse. Eilige gleichzeitige Events werden nicht gesperrt.

0294 Priorität lfd. Event

029A *********************************** KL EVENT ENABLE

Normale gleichzeitge Ereignisse zulassen.

029A Priorität lfd. Event

02A0 *** KL LOG EXT

Residente Erweiterungen einhängen.

02B1 ********************************** KL FIND COMMAND

Befehl in allen eingehängten Speicherbereichen suchen.

02B1	auszuführender Befehl
02B7	(0553) KL ROM OFF & KOFIGURATION SAVE
02DA	(0524) KL PROBE ROM CONT'D
02E4	MC START PROGRAM
02FC	(051F) KL ROM SELECT CONT'D
0307	auszuführender Befehl
0323	(052D) KL ROM DESELECT CONT'D

0326 ** KL ROM WALK

Findet und initialisiert ROM-Erweiterungen, damit diese ROMs verfügbar sind.

0328 KL INIT BACK

0330 ** KL INIT BACK

ROM-Erweiterungen einhängen.

0330 lfd. ROM-Konfiguration
0339 (051F) KL ROM SELECT CONT'D
0351 (lfd. Exp.-ROM)
0360 KL LOG EXT
0366 (052D) KL ROM DESELECT CONT'D

0379 ** Add Event

0388 ** Delete Event

0397 ****************** KL RAM-KONFIGURATION SETZEN

Hier wird eine Umschaltung zwischen den verschiedenen RAM-Konfigurationen des CPC 6128 vorgenommen.

0398 Rette Register
0399 lfd. RAM-Konfiguration
039E Aufbereitung für Gate-Array
03A0 Umschaltung RAM-Konfiguration
03A3 alter Registerzustand wiederherstellen

03A6 (0505) KL U ROM ENABLE CONT'D
03A9 (050C) KL U ROM DISABLE CONT'D
03AC (04F7) KL L ROM ENABLE CONT'D
03AF (04FE) KL L ROM DISABLE CONT'D
03B2 (0516) KL ROM RESTORE CONT'D
03B5 (051F) KL ROM SELECT CONT'D
03B8 (0543) KL CURR SELECTION CONT'D
03BB (0524) KL PROBE ROM CONT'D

03BE (052D) KL ROM DESELECT CONT'D
03C1 (0547) KL LDIR CONT'D
03C4 (054D) KL LDDR CONT'D

03C7 ****************************** KL POLL SYNCHRONOUS

Gibt es einen Event mit höherer Priorität als die des laufenden?

03D6 (Start Sync Pending Queue)
03E0 (Priorität lfd. Event)

03E7 ****************** RST 7 INTERRUPT ENTRY CONT'D

Vergleichen Sie mit RST 7 INTERRUPT ENTRY.

03E9 KL EXT INTERRUPT ENTRY
03F4 L ROM enable
03F6 Scan Events
03FE (diverse Flags für Int. Routine)
0418 alte Konfiguration setzen

041E ************************** KL EXT INTERRUPT ENTRY

0423 L ROM disable

042A ******************************** KL LOW PCHL CONT'D

Sprung ins untere ROM oder RAM.

0430 ****************************** RST 1 LOW JUMP CONT'D

Vergleichen Sie mit RST 1 LOW JUMP.

043C Akku viermal nach links rotieren
0445 (0456) Konfiguration vorbereiten und Sprung ausführen
0456 Sprungadresse auf den Stack legen
0458 ROM Konfiguration setzen
045E vorbereiteten Sprung ausführen

045F ********************************* KL FAR PCHL CONT'D

0467 ********************************* KL FAR ICALL CONT'D

046D ************************* RST 3 LOW FAR CALL CONT'D

Vergleichen Sie mit RST 3 LOW FAR CALL.

047C ROM# > 252?
047E ja springe
0480 Expansion-ROM
0482 einschalten
0484 lfd. Expansion ROM
04A2 L ROM disable
04A4 U ROM enable
04A6 (0456) Konfiguration vorbereiten und Sprung ausführen
04AF alte
04B0 Expansion-ROM-
04B1 Konfiguration
04B3 wiederherstellen
04B5 (lfd. Expansion-ROM)

04BD ********************************* KL SIDE PCHL CONT'D

04C3 ************************* RST 2 LOW SIDE CALL CONT'D

Vergleichen Sie mit RST 2 LOW SIDE CALL.

04D5 (lfd. ROM-Konfiguration)

04DB ****************************** RST 5 FIRM JUMP CONT'D

Vergleichen Sie mit RST 5 FIRM JUMP.

04E3 L ROM enable
04E5 Sprungadresse laden

04EB und Sprung ausführen
04F0 L ROM disable

04F7 ************************* KL L ROM ENABLE CONT'D

Unteres ROM einschalten.

04FA L ROM enable
04FC Sprung zur Durchführung

04FE ************************* KL L ROM DISABLE CONT'D

Unteres ROM ausschalten.

0501 L ROM disable
0503 Sprung zur Durchführung

0505 ************************* KL U ROM ENABLE CONT'D

Oberes ROM einschalten.

0508 U ROM enable
050A Sprung zur Durchführung

050C ************************* KL U ROM DISABLE CONT'D

Oberes ROM ausschalten.

050F U ROM disable
0511 Durchführung

0516 *************************** KL ROM RESTORE CONT'D

Alte ROM-Konfiguration wiederherstellen.

0517 a enthält
0518 die alte
0519 Konfiguration
051D Sprung zur Durchführung

051F ***************************** KL ROM SELECT CONT'D

Ein bestimmtes oberes ROM auswählen.

051F (0505) KL U ROM ENABLE CONT'D

0524 ****************************** KL PROBE ROM CONT'D

ROM untersuchen.

0524 (051F) KL ROM SELECT CONT'D

052D ************************** KL ROM DESELECT CONT'D

Alte obere ROM-Konfiguration wiederherstellen.

052F (0516) KL ROM RESTORE CONT'D
0535 Expansion-ROM (# in c)
0537 einschalten
0539 lfd. Expansion-ROM

0543 ************************ KL CURR SELECTION CONT'D

Welches obere ROM ist an?

0543 lfd. Expansion-ROM

0547 *************************************** KL LDIR CONT'D

LDIR bei blockierten ROMs.

0547 (0553) KL ROM OFF & KONFIG. SAVE

054D ************************************* KL LDDR CONT'D

LDDR bei blockierten ROMs.

054D (0553) KL ROM OFF & KONFIG. SAVE

0553 ********************** KL ROM OFF & KONFIG. SAVE

0555	RET-Adresse manipulieren
0556	alte Konfiguration auf Stack sichern
0557	ROMs
0559	disable
055D	call (hl)
0561	alte
0562	Konfiguration
0563	wiederherstellen
0568	RET-Adresse manipulieren

056C **************************** RST 4 RAM LAM CONT'D

Vergleichen Sie RST 4 RAM LAM.

056F	ROMs
0571	disable
0576	Byte holen
0578	alte Konfiguration setzen

057D ************************************ KL RAM LAM (IX)

Entspricht ld a,(ix).

057F	ROMs
0581	disable
0583	Byte holen
0586	alte Konfiguration setzen

2.5.2 Maschine Pack (MC)

Kommen wir nun zum maschinennahen Teil des Betriebssystems. Hier werden die diversen Schnittstellen und Peripheriebausteine wie PIO und PSG bedient. Dieses Verfahren hat den Vorteil, daß bei eventuellen Änderungen der Hardware nur das MASCHINE PACK angepaßt werden muß, vergleichbar dem BIOS im CP/M.

0591 ** Reset Cont'd

0592 Control
0597 Port A
059C Port C
05A1 Centronics
05A6 Port B
05AA LK4 isolieren
05AC Ende Tabelle 60Hz
05AF 50Hz? Springe wenn nicht
05B1 Ende Tabelle 50Hz
05B7 Video Register-Adresse laden
05BC Video Register laden

05C5 ** Tabelle 60Hz

```
3F 28 2E 8E 26 00 19 1E
00 07 00 00 30 00 C0 00
```

05D5 ** Tabelle 50Hz

```
3F 28 2E 8E 1F 06 19 1B
00 07 00 00 30 00 C0 00
```

05E5 Kaltstart
05E8 in Fortsetzungs-
05EB adresse

05ED ********************************** MC BOOT PROGRAM

Setzt das Betriebssystem zurück und übergibt die Steuerung einer Routine in (hl).

05F1	SOUND RESET
05F5	Peripherie
05F8	rücksetzen
05FA	KL CHOKE OFF
0601	KM RESET
0604	TXT RESET
0607	SCR RESET
060A	KL U ROM ENABLE CONT'D
060E	jp (hl)
0613	MC START PROGRAM
0617	Ladefehler

061C ********************************* MC START PROGRAM

System vollständig Initialisieren und Aufruf des Programms dessen Startadresse in hl steht.

061C	trifft nach 066F auf RET
0620	setze Interrupt-Modus 1
0622	rette Registerinhalte
0623	Palette Pointer reset
0628	eventuell angeschlossene
062B	Peripherie reset
062D	RAM-Konfiguration
0630	zurücksetzen
0632	Floppy-Motor on/off Flip/Flop
0636	Floppy-Motor ausschalten
0638	&7f9 Bytes von Start-
063B	adresse &B100 zur Ziel-
063E	adresse &B101 kopieren
0641	lade &B100 mit Inhalt von Akku
0642	führe Kopiervorgang durch
0644	U ROM off & L ROM on

0647 Sreen Mode 1
0649 stelle alte Registerinhalte wieder her
0652 Restore High Kernel Jumps
0655 JUMP RESTORE
0658 KM INITIALISE
065B SOUND RESET
065E SCR INITIALISE
0661 TXT INITIALISE
0664 GRA INITIALISE
0667 CAS INITIALISE
066A MC RESET PRINTER
066F jp (hl)
0674 U ROM initialisieren

0677 ** Kaltstart

067A TXT SET CUSOR
067D Firmennamen ausgeben
0680 Meldungen ausgeben
0683 Einschaltmeldung
0686 Meldungen ausgeben

1672 0688 ************************************** Einschaltmeldung

1673 0689 128K
068E Microcomputer
069D (v3)
06A4 Copyright
06B0 c1985
06B6 Amstrad
06BE Consumer
06C7 Electronics
06D3 plc
06D9 and
06DD Locomotive
06E8 Software
06F1 Ltd

06F9 Ladefehler-Meldung

06FC ** Meldung ausgeben

0700 TXT OUTPUT
0703 Meldung ausgeben

0705 *************************************** Ladefehler-Meldung

0705 ***
0709 PROGRAM
0711 LOAD
0716 FAILED
071E ***

0725 Port B
0728 LK1...3 isolieren
072A /2
072B Firmennamen

0738 *** Firmennamen

0738 Arnold
073F Amstrad
0747 Orion
074D Schneider
0757 Awa
075B Solavox
0763 Saisho
076A Triumph
0772 Isp

0776 ** MC SET MODE

Bildschirmmodus setzen.

0776	Mode>2?
0778	wenn ja Rücksprung
077B	Mode Bits
077D	rücksetzen
0780	neuen Mode
0781	setzen

0786 ************************************** MC CLEAR INKS

Bildschirmrand und alle Inks auf eine Farbe setzen.

0786	Inhalt von hl auf den Stack legen
0787	dann hl mit &0000 laden
078A	sechs Bytes weiter

078C ** MC SET INKS

Farben aller Inks und des Bildschirmrandes ausgeben.

078C	Inhalt von hl auf den Stack legen
078D	dann hl mit &0001 laden
0793	Border Farbe
0796	Farbe ausgeben
079A	Adresse Ink 0
079C	Farbe ausgeben
07A4	alle Farbspeicher laden

07AA *** Farbe ausgeben

07AA	Palette Pointer
07AD	Bit 5,6 und 7 des Akkus löschen
07AF	dann Bit 6 setzen
07B1	Farbe

07B4 ************************************ MC WAIT FLYBACK

Strahlrücklauf abwarten.

07B6 Port B
07BA VSYNC?
07BB wenn nicht warten

07C0 ************************************ MC SCREEN OFFSET

Setze den Bildschirm-Offset.

07C3 Alle Bits außer 4 und 5 löschen
07C8 alle Bits außer 0 und 1 löschen
07CE Video Contr Register 12
07D1 Bildschirm Start Hi
07D5 Register 13
07DC Bildschirm Start Lo

07E0 ************************************ MC RESET PRINTER

Indirekten Verzweigungspunkt für Drucker zurücksetzen.

07E0 Startadresse
07E3 Zieladresse
07E6 21 Bytes
07E9 kopieren
07EE Move (hl+3) nach ((hl+1)),cnt = (hl)
07F1 db 03 3 Bytes
07F2 dw BDF1 Zieladresse
07F4 MC WAIT PRINTER

07F7 ************************************ Umlaute konvertieren

Die folgende Tabelle wurde von MC RESET PRINTER ins RAM kopiert (Zieladresse &B804). Das erste Byte der Tabelle gibt die Länge der Tabelle in Bytes an. Dann folgen mehrere Byte-Pärchen, von denen das erste jeweils den internen Tastaturcode,

das zweite das standardmäßig zugeordnete Zeichen angibt. Wird nun diese Tabelle im RAM geändert, so ist es möglich, die den Tastaturcodes zugeordneten Zeichen zu manipulieren und so beispielsweise eine deutsche Tastatur zu erzeugen.

```
07F7    db 0A  Anzahl Bytes

07F8    db A0  interner Tastaturcode
07F9    db 5E  zugeordnetes Zeichen  ^

07FA    db A1  interner Tastaturcode
07FB    db 5C  zugeordnetes Zeichen  \

07FC    db A2  interner Tastaturcode
07FD    db 7B  zugeordnetes Zeichen  {

07FE    db A3  interner Tastaturcode
07FF    db 23  zugeordnetes Zeichen  #

0800    db A6  interner Tastaturcode
0801    db 40  zugeordnetes Zeichen  @

0802    db AB  interner Tastaturcode
0803    db 7C  zugeordnetes Zeichen  |

0804    db AC  interner Tastaturcode
0805    db 7D  zugeordnetes Zeichen  }

0806    db AD  interner Tastaturcode
0807    db 7E  zugeordnetes Zeichen  ~

0808    db AE  interner Tastaturcode
0809    db 5D  zugeordnetes Zeichen  ]

080A    db AF  interner Tastaturcode
080B    db 5B  zugeordnetes Zeichen  [
```

080C **************************** MC ZEICHENZUORDNUNG

Hier erfolgt die Manipulation von Umlaute konvertieren.

080C hl: Startadresse der neuen Zeichentabelle (RAM)
0812 Umlaute konvertieren (RAM)
0817 KL LDIR CONT'D

081B ** MC PRINT CHAR

Gibt das Zeichen in a auf den Centronics-Port aus. Nach Rückkehr aus dieser Routine ist carry gesetzt, wenn das Zeichen erfolgreich gegesetzt wurde.

0826 Umlaut?
0828 springe wenn nicht
082F MC WAIT PRINTER

0835 ************************************** MC WAIT PRINTER

Sende ein Zeichen an den Drucker; wenn dieser nicht bereit ist, warte eine Zeitperiode.

0838 MC BUSY PRINTER
083B MC SEND PRINTER

0844 ************************************** MC SEND PRINTER

Sendet ein Zeichen an den Drucker, der nicht busy sein darf.

0847 Byte ohne Strobe
0849 an Drucker
084E Strobe Ein
0853 Strobe Aus

0858 ************************************ MC BUSY PRINTER

Untersuche, ob der Drucker beschäftigt ist.

085A	Port B
085E	Drucker Busy
085F	nach Carry

0863 ********************************** MC SOUND REGISTER

Sound Controller mit Daten versorgen. MC SOUND REGISTER ist für den Musikfan interessant. Ohne daß Sie sich mit der relativ komplizierten Datenübergabe an den PSG plagen müssen, brauchen Sie nur im Akku die gewünschte Registernummer und in c das Datenbyte zu übergeben.

0864	Port A
0866	Sound Register#
0868	Port C
086A	Sound Chip
086C	auf Eingabe
086E	& Strobe Ein
0872	Strobe Aus
0874	Port A
0876	Sound Daten
0878	Port C
087D	Daten
087F	einlatchen

0883 ** Scan Keyboard

0883	Port A
0886	Sound Register 14 (Keyboard X Input)
0888	Port C
0891	Strobe Ein
0893	Strobe Aus
0896	Port A&B = Input
0898	Control

089D	Port C
089F	Keyboard Y Output und X Input
08A1	Port A
08A3	Daten (Keyboard X Input) nach Akku
08AC	Keyboard Y+1
08B0	alle Y-Leitungen bearbeitet?
08B2	nein dann nächste Leitung
08B5	Port A Output
08B7	Control
08BA	Port C

2.5.3 Jump Restore (JRE)

Dieses Pack dient ausschließlich dazu, die **MAIN-JUMP**-Adressen wieder auf ihre Default-Werte zu setzen. Dabei wird den **FIRM JUMPS** ein RST 1 vorangestellt, den **ARITHMETIK JUMPS** ein RST 5.

Wenn Sie nach heftigem Programmieren der Meinung sind, allzu viele Vektoren verbogen zu haben, so ziehen Sie einfach die 'Notbremse', indem Sie diesen **JUMP RESTORE** anspringen. Das ist auch immer dann ratsam, wenn Sie ein Programm verlassen, in dem Sie viele Routinen des Betriebssystems durch eigene ersetzt haben.

08BD ** JUMP RESTORE

08BD	Main Jump Adress
08C0	Zeiger auf Vektorbereich im RAM
08C3	b: Anzahl der Vektoren c: Code RST 1
08C6	Kopiere Vektortabelle
08C9	b: Anzahl der Vektoren c: Code RST 5
08CD	Code RST speichern
08CE	Zeiger+1 (RAM)
08CF	Ein Byte vom ROM ins RAM
08D1	bc auf Wert vor LDI
08D2	Akku komplementieren
08D3	Bit 5 nach
08D4	Bit 7 schieben
08D5	und isolieren
08D7	Bit 0-6 von Adresse High-Byte holen
08D8	High-Byte speichern
08D9	Zeiger+1 (RAM)
08DA	Zeiger+1 (ROM)
08DB	weiter solange nötig
08DD	Rücksprung aus dem Unterprogramm

08DE ************************************** Main Jump Adress

```
08DE    dw 1B5C KM INITIALISE
08E0    dw 1B98 KM RESET
08E2    dw 1BBF KM WAIT CHAR
08E4    dw 1BC5 KM READ CHAR
08E6    dw 1BFA KM CHAR RETURN
08E8    dw 1C46 KM SET EXPAND
08EA    dw 1CB3 KM GET EXPAND
08EC    dw 1C04 KM EXPAND BUFFER
08EE    dw 1CDB KM WAIT KEY
08F0    dw 1CE1 KM READ KEY
08F2    dw 1E45 KM TEST KEY
08F4    dw 1D38 KM GET STATE
08F6    dw 1DE5 KM GET JOYSTICK
08F8    dw 1ED8 KM SET TRANSLATE
08FA    dw 1EC4 KM GET TRANSLATE
08FC    dw 1EDD KM SET SHIFT
08FE    dw 1EC9 KM GET SHIFT
0900    dw 1EE2 KM SET CONTROL
0902    dw 1ECE KM GET CONTROL
0904    dw 1E34 KM SET REPEAT
0906    dw 1E2F KM GET REPEAT
0908    dw 1DF6 KM SET DELAY
090A    dw 1DF2 KM GET DELAY
090C    dw 1DFA KM ARM BREAK
090E    dw 1E0B KM DISARM BREAK
0910    dw 1E19 KM BREAK EVENT

0912    dw 1074 TXT INITIALISE
0914    dw 1984 TXT RESET
0916    dw 1459 TXT VDU ENABLE
0918    dw 1452 TXT VDU DISABLE
091A    dw 13FE TXT OUTPUT
091C    dw 1335 TXT WR CHAR
091E    dw 13AC TXT RD CHAR
0920    dw 13A8 TXT SET GRAPHIC
0922    dw 1208 TXT WIN ENABLE
```

```
0924    dw 1252 TXT GET WINDOW
0926    dw 154F TXT CLEAR WINDOW
0928    dw 115A TXT SET COLUMN
092A    dw 1165 TXT SET ROW
092C    dw 1170 TXT SET CURSOR
092E    dw 117C TXT GET CURSOR
0930    dw 1286 TXT CUR ENABLE
0932    dw 1297 TXT CUR DISABLE
0934    dw 1276 TXT CUR ON
0936    dw 127E TXT CUR OFF
0938    dw 11CA TXT VALIDATE
093A    dw 1265 TXT PLACE/REMOVE CURSOR
093C    dw 1265 TXT PLACE/REMOVE CURSOR
093E    dw 12A6 TXT SET PEN
0940    dw 12BA TXT GET PEN
0942    dw 12AB TXT SET PAPER
0944    dw 12C0 TXT GET PAPER
0946    dw 12C6 TXT INVERSE
0948    dw 137B TXT SET BACK
094A    dw 1388 TXT GET BACK
094C    dw 12D4 TXT GET MATRIX
094E    dw 12F2 TXT SET MATRIX
0950    dw 12FE TXT SET M TABLE
0952    dw 132B TXT GET M TABLE
0954    dw 14D4 TXT GET CONTROLS
0956    dw 10E4 TXT STR SELECT
0958    dw 1103 TXT SWAP STREAMS

095A    dw 15A8 GRA INITIALISE
095C    dw 15D7 GRA RESET
095E    dw 15FE GRA MOVE ABSOLUTE
0960    dw 15FB GRA MOVE RELATIVE
0962    dw 1606 GRA ASK CURSOR
0964    dw 160E GRA SET ORIGIN
0966    dw 161C GRA GET ORIGIN
0968    dw 16A5 GRA WIN WIDTH
096A    dw 16EA GRA WIN HEIGHT
096C    dw 1717 GRA GET W WIDTH
096E    dw 172D GRA GET W HEIGHT
```

```
0970    dw 1736 GRA CLEAR WINDOW
0972    dw 1767 GRA SET PEN
0974    dw 1775 GRA GET PEN
0976    dw 176E GRA SET PAPER
0978    dw 177A GRA GET PAPER
097A    dw 1783 GRA PLOT ABSOLUTE
097C    dw 1780 GRA PLOT RELATIVE
097E    dw 1797 GRA TEST ABSOLUTE
0980    dw 1794 GRA TEST RELATIVE
0982    dw 17A9 GRA LINE ABSOLUTE
0984    dw 17A6 GRA LINE RELATIVE
0986    dw 1940 GRA WR CHAR

0988    dw 0ABF SCR INITIALISE
098A    dw 0AD0 SCR RESET
098C    dw 0B37 SCR SET OFFSET
098E    dw 0B3C SCR SET BASE
0990    dw 0B56 SCR GET LOCATION
0992    dw 0AE9 SCR SET MODE
0994    dw 0B0C SCR GET MODE
0996    dw 0B17 SCR MODE CLEAR
0998    dw 0B5D SCR CHAR LIMITS
099A    dw 0B6A SCR CHAR POSITION
099C    dw 0BAF SCR DOT POSITION
099E    dw 0C05 SCR NEXT BYTE
09A0    dw 0C11 SCR PREV BYTE
09A2    dw 0C1F SCR NEXT LINE
09A4    dw 0C39 SCR PREV LINE
09A6    dw 0C8E SCR INK ENCODE
09A8    dw 0CA7 SCR INK DECODE
09AA    dw 0CF2 SCR SET INK
09AC    dw 0D1A SCR GET INK
09AE    dw 0CF7 SCR SET BORDER
09B0    dw 0D1F SCR GET BORDER
09B2    dw 0CEA SCR SET FLASHING
09B4    dw 0CEE SCR GET FLASHING
09B6    dw 0DB9 SCR FILL BOX
09B8    dw 0DBD SCR FLOOD BOX
09BA    dw 0DE5 SCR CHAR INVERT
```

```
09BC    dw 0E00 SCR HW ROLL
09BE    dw 0E44 SCR SW ROLL
09C0    dw 0EF9 SCR UNPACK
09C2    dw 0F2A SCR REPACK
09C4    dw 0C55 SCR ACCESS
09C6    dw 0C74 SCR PIXELS
09C8    dw 0F93 SCR HORIZONTAL
09CA    dw 0F9B SCR VERTICAL

09CC    dw 24BC CAS INITIALISE
09CE    dw 24CE CAS SET SPEED
09D0    dw 24E1 CAS NOISY
09D2    dw 2BBB CAS START MOTOR
09D4    dw 2BBF CAS STOP MOTOR
09D6    dw 2BC1 CAS RESTORE MOTOR
09D8    dw 24E5 CAS IN OPEN
09DA    dw 2550 CAS IN CLOSE
09DC    dw 2557 CAS IN ABANDON
09DE    dw 25A0 CAS IN CHAR
09E0    dw 2618 CAS IN DIRECT
09E2    dw 2607 CAS RETURN
09E4    dw 2603 CAS TEST EOF
09E6    dw 24FE CAS OUT OPEN
09E8    dw 257F CAS OUT CLOSE
09EA    dw 2599 CAS OUT ABANDON
09EC    dw 25C6 CAS OUT CHAR
09EE    dw 2653 CAS OUT DIRECT
09F0    dw 2692 CAS CATALOG
09F2    dw 29AF CAS WRITE
09F4    dw 29A6 CAS READ
09F6    dw 29C1 CAS CHECK

09F8    dw 1FE9 SOUND RESET
09FA    dw 2114 SOUND QUEUE
09FC    dw 21CE SOUND CHECK
09FE    dw 21EB SOUND ARM EVENT
0A00    dw 21AC SOUND RELEASE
0A02    dw 2050 SOUND HOLD
0A04    dw 206B SOUND CONTINUE
```

```
0A06    dw 2495 SOUND AMPL ENVELOPE
0A08    dw 249A SOUND TONE ENVELOPE
0A0A    dw 24A6 SOUND A ADRESS
0A0C    dw 24AB SOUND T ADRESS

0A0E    dw 005C KL CHOKE OFF
0A10    dw 0326 KL ROM WALK
0A12    dw 0330 KL INIT BACK
0A14    dw 02A0 KL LOG EXT
0A16    dw 02B1 KL FIND COMMAND
0A18    dw 0163 KL NEW FRAME FLY
0A1A    dw 016A KL ADD FRAME FLY
0A1C    dw 0170 KL DEL FRAME FLY
0A1E    dw 0176 KL NEW FAST TICKER
0A20    dw 017D KL ADD FAST TICKER
0A22    dw 0183 KL DEL FAST TICKER
0A24    dw 01B3 KL ADD TICKER
0A26    dw 01C5 KL DEL TICKER
0A28    dw 01D2 KL INIT EVENT
0A2A    dw 01E2 KL EVENT
0A2C    dw 0227 KL SYNC RESET
0A2E    dw 0284 KL DELETE SYNCHRONOUS
0A30    dw 0255 KL NEXT SYNC
0A32    dw 0219 KL DO SYNC
0A34    dw 0276 KL DONE SYNC
0A36    dw 0294 KL EVENT DISABLE
0A38    dw 029A KL EVENT ENABLE
0A3A    dw 028D KL DISARM EVENT
0A3C    dw 0099 KL TIME PLEASE
0A3E    dw 00A3 KL TIME SET

0A40    dw 05ED MC BOOT PROGRAM
0A42    dw 061C MC START PROGRAM
0A44    dw 07B4 MC WAIT FLYBACK
0A46    dw 0776 MC SET MODE
0A48    dw 07C0 MC SCREEN OFFSET
0A4A    dw 0786 MC CLEAR INKS
0A4C    dw 078C MC SET INKS
0A4E    dw 07E0 MC RESET PRINTER
```

```
0A50    dw 081B MC PRINT CHAR
0A52    dw 0858 MC BUSY PRINTER
0A54    dw 0844 MC SEND PRINTER
0A56    dw 0863 MC SOUND REGISTER

0A58    dw 08BD JUMP RESTORE

0A5A    dw 1D3C KM SET STATE
0A5C    dw 1BFE KM PUFFER ENTLEEREN
0A5E    dw 1460 TXT LFD. CURSOR FLAG NACH AKKU
0A60    dw 15EC GRA NN
0A62    dw 19D5 GRA PARAM RETTEN
0A64    dw 17B0 GRA MASK PARAM RETTEN
0A66    dw 17AC GRA MASK PARAM RETTEN
0A68    dw 1624 GRA KOORD. KONVERTIEREN
0A6A    dw 19D9 GRA FILL
0A6C    dw 0B45 SCR VERÄNDERUNG SCREEN START
0A6E    dw 080C MC ZEICHENZUORDNUNG
0A70    dw 0397 KL RAM-KONFIGURATION SETZEN

0A72 ************************************** BASIC Jump Adr.

0A72    dw 2C02 EDIT

0A74    dw 2F91 FLO VARIAB. VON (DE) NACH (HL) KOP.
0A76    dw 2F9F FLO INTEGER NACH FLIESSKOMMA
0A78    dw 2FC8 FLO 4-BYTE-WERT NACH FLO
0A7A    dw 2FD9 FLO FLO NACH INT
0A7C    dw 3001 FLO FLO NACH INT
0A7E    dw 3014 FLO FIX
0A80    dw 3055 FLO INT
0A82    dw 305F FLO
0A84    dw 30C6 FLO ZAHL MIT 10^A MULITIPLIZIEREN
0A86    dw 34A2 FLO ADDITION
0A88    dw 3159 FLO RND
0A8A    dw 349E FLO SUBTRAKTION
0A8C    dw 3577 FLO MULTIPLIKATION
0A8E    dw 3604 FLO DIVISION
0A90    dw 3188 FLO LETZTEN RND-WERT HOLEN
```

```
0A92   dw 36DF FLO VERGLEICH
0A94   dw 3731 FLO VORZEICHENWECHSEL
0A96   dw 3727 FLO SGN
0A98   dw 3345 FLO DEG/RAD
0A9A   dw 2F73 FLO PI
0A9C   dw 32AC FLO SQR
0A9E   dw 32AF FLO POTENZIERUNG
0AA0   dw 31B6 FLO LOG
0AA2   dw 31B1 FLO LOG10
0AA4   dw 322F FLO EXP
0AA6   dw 3353 FLO SIN
0AA8   dw 3349 FLO COS
0AAA   dw 33C8 FLO TAN
0AAC   dw 33D8 FLO ATN
0AAE   dw 2FD1 FLO 4-BYTE-WERT NACH FLO
0AB0   dw 3136 FLO RND INIT
0AB2   dw 3143 FLO SET RND SEED

0AB4 ************************ Move (hl+3) nach ((hl+1)),cnt=(hl)
```

2.5.4 Screen Pack (SCR)

Das **SCREEN PACK** ist dem **TEXT-** und dem **GRAPHICS PACK** untergeordnet. Es ist praktisch die Exekutive für diese beiden Packs und damit für die unmittelbare Handhabung des Bildschirms zuständig.

0ABF ************************************** SCR INITIALISE

Vollständige Initialisierung des Screen-Packs.

0ABF Default Farben
0AC2 MC CLEAR INKS
0AC7 (High Byte Screen Start)
0ACA SCR RESET

0AD0 ** SCR RESET

Rücksetzen des Screen-Packs.

0AD1 SCR ACCESS
0AD4 Restore SCR Indirections
0AD7 Move (hl+3) nach ((hl+1)),cnt=(hl)
0ADA Reset Farben
0ADD db 09 9 Bytes
0ADE dw BDE5 Zieladresse
0AE0 SCR READ
0AE3 SCR WRITE
0AE6 SCR CLEAR

0AE9 ************************************** SCR SET MODE

Bildschirm in einen neuen Modus setzen.

0AFF SCR CLEAR

0B0C ** SCR GET MODE

Gewärtigen Bildschirm-Modus holen.

0B0C (curr. Screen Mode)

0B17 ** SCR CLEAR

Bildschirm löschen.

0B1D SCR SET OFFSET
0B25 hl=Basis Adresse
0B26 de=Basis Adresse+1
0B28 16k
0B2C Bildschirm löschen

0B31 (curr. Screen Mode)
0B34 MC SET MODE

0B37 *************************************** SCR SET OFFSET

Startadresse des ersten Zeichens relativ zur Basisadresse des Video-RAMs setzen.

0B37 (High Byte Screen Start)

0B3C *** SCR SET BASE

Basisadresse des Video-RAMs.

0B3C (Position innerhalb einer Zeile)
0B42 MC SCREEN OFFSET

0B45 **************** SCR VERÄNDERUNG SCREEN START

0B47 (High Byte Screen Start)
0B51 (Position innerhalb einer Zeile)

0B56 ********************************** SCR GET LOCATION

Lfd. Bildschirmstart? (Basis+Offset)

0B56 (Position innerhalb einer Zeile)
0B59 (High Byte Screen Start)

0B5D ************************************* SCR CHAR LIMITS

Größtmögliche Zeilen- und Spaltenzahl des Bildschirms holen (abhängig vom Modus).

0B5D SCR GET MODE

0B6A ********************************** SCR CHAR POSTION

Übersetze phys. Koordinaten in eine Bildschirmposition.

0B6B SCR GET MODE
0B93 (High Byte Screen Start)
0BA6 SCR CHAR POSITION

0BAF *********************************** SCR DOT POSITION

Bildschirmposition für ein Pixel ermitteln.

0BED (High Byte Screen Start)

0BF6 SCR GET MODE

0C05 *************************************** SCR NEXT BYTE

Liefert in hl die Bildschirmadresse der nächsten Byteposition zurück, wenn Sie vor dem Ansprung hl mit der alten Adresse versorgt haben. So überflüssig das scheinen mag, so praktisch ist es. Es ist nämlich, aufgrund der auf Graphikbetrieb ausgerichteten Organisation des Bildschirms, nicht einfach, die Byteposition zu ermitteln. Zudem ist die Distanz vom Modus abhängig.

Beachten Sie, daß, wenn die nächste Position nicht mehr innerhalb des Bildschirms läge, die zurückgelieferte Adresse unsinnig ist. Sie liegt dann im Bereich der letzten (für die Darstellung unbenutzten) Bytes des Video-RAMs.

0C11 ** SCR PREV BYTE

Liefert in hl die Bildschirmadresse der vorigen Byteposition zurück, wenn Sie vor dem Ansprung hl mit der alten Adresse versorgt haben. Vergleichen Sie mit SCR NEXT BYTE.

0C1F ** SCR NEXT LINE

Arbeitet analog zu SCR NEXT BYTE, nur daß die Bildschirmadresse um eine ganze Zeile vorgerechnet wird. Auch hier ist die Adresse beim Verlassen des darstellbaren Bereiches ungültig.

0C39 *** SCR PREV LINE

Arbeitet analog zu SCR PREV BYTE, nur daß die Bildschirmadresse um eine ganze Zeile zurückgerechnet wird. Vergleichen Sie mit SCR NEXT LINE und SCR PREV BYTE.

0C55 *** SCR ACCESS

Steuerzeichen sichtbar/unsichtbar setzen.

0C57	SCR PIXELS (FORCE MODE)
0C5E	Low Byte XOR Mode
0C62	Low Byte AND Mode
0C66	Low Byte OR Mode
0C68	jp
0C6A	(Write Indirection)

0C71 ** SCR WRITE

0C71 Write Indirection

0C74 **************************** SCR PIXELS (FORCE Mode)

Punkt auf den Bildschirm setzen.

0C7A ** XOR Mode

0C7F ** AND Mode

0C85 ** OR Mode

0C8A ** SCR READ

0C8E ************************************ SCR INK ENCODE

Codieren einer Ink, so daß alle Bildpunkte auf diese Ink gesetzt werden.

0CA7 ************************************* SCR INK DECODE

Entschlüsseln einer Ink.

0CC9 SCR GET MODE

0CD8 *** Reset Farben

0CD8 Default Farben
0CDB Farbspeicher 2. Farben
0CE4 (Flag lfd Farbsatz)

0CEA *********************************** SCR SET FLASHING

Blinkzeiten zur Farbdarstellung für alle Inks und den Rahmen setzen.

0CEA (Flash Periods)

0CEE ********************************** SCR GET FLASHING

Blinkzeiten ermitteln (Inks und Rahmen).

0CEE (Flash Periods)

0CF2 *** SCR SET INK

Zuordnung der beiden Farben, die zur Darstellung einer Ink verwendet werden.

0CF5 Set Colour

0CF7 ************************************** SCR SET BORDER

Zuordnung der beiden Farben, die zur Darstellung eines Rahmens verwendet werden.

0CF8 ** Set Colour

0CFA Farbmatrix Eintrag holen
0CFF Farbmatrix Eintrag holen
0D04 Ink Adresse holen

0D10 ********************************* Farbmatrix Eintrag holen

0D1A *** SCR GET INK

Holen der beiden Farben, die zur Darstellung einer Ink verwendet werden.

0D1D Get Colour

0D1F ************************************* SCR GET BORDER

Holen der beiden Farben, die zur Darstellung eines Rahmens verwendet werden.

0D20 ** Get Colour

0D20 Ink Adresse holen
0D2C Farbmatrix

0D35 ** Ink Adresse holen

0D38 Farbspeicher 1. Farben
0D42 Event Block: Set Inks
0D46 KL DEL FRAME FLY
0D49 Flash Inks
0D4C Set Inks on Frame Fly
0D52 KL NEW FRAME FLY
0D55 Event Block: Set Inks
0D58 KL DEL FRAME FLY
0D5B Params d. lfd. Frabsatz holen
0D5E MC CLEAR INKS

0D61 ************************************ Set Inks on Frame Fly

0D61 curr. Flash Period
0D65 Flash Inks
0D6B Params d. lfd Farbsatz holen
0D6E MC SET INKS

0D73 *** Flash Inks

0D73 Params d. lfd Farbsatz holen
0D76 (curr. Flash Period)
0D79 MC SET INKS
0D7C Flag lfd. Farbsatz

0D87 ****************************** Params d. lfd Farbsatz holen

0D87 Farbspeicher 1. Farben
0D8A (Flag lfd. Farbsatz)

0D8E (Flash Period 1. Colour)
0D92 Farbspeicher 2. Farben
0D95 (Flash Periods)

0D99 ** Farbmatrix

0D99 14 04 15 1C 18 1D 0C 05
0DA1 0D 16 06 17 1E 00 1F 0E
0DA9 07 0F 12 02 13 1A 19 1B
0DB1 0A 03 0B 01 08 09 10 11

0DB9 *** SCR FILL BOX

Vorgegebenes Fenster mit einer Farbe füllen (Positionen zeichenbezogen, Mode-abhängig).

0DBD ************************************** SCR FLOOD BOX

Vorgegebenes Fenster mit einer Farbe füllen (Positionen sind Bildschirmadressen, Mode-unabhängig).

0DC6 SCR NEXT BYTE
0DDE SCR NEXT LINE
0DE2 SCR FLOOD BOX

0DE5 ************************************* SCR CHAR INVERT

Bei einem Zeichen Vorder- und Hintergrundfarbe vertauschen.

0DE8 SCR CHAR POSITION
0DF2 SCR NEXT BYTE

0DF8 ********************************* Farbspeicher adressieren

0DF9 SCR NEXT LINE

0E00 ** SCR HW ROLL

Schiebt den Bildschirm (hardwaremäßig) um eine Zeile nach unten, wenn b=0 ist, und um eine Zeile nach oben, benn b<> ist. Im Akku muß der Wert für die Farbe stehen, die die neue (leere) Zeile annehmen soll.

0E0B	MC WAIT FLYBACK
0E32	(High Byte Screen Start)
0E3A	SCR FLOOD BOX
0E41	SCR SET OFFSET

0E44 ** SCR SW ROLL

Verschiebt einen Bildschirmbereich softwaremäßig. a und b sind wie bei SCR HW ROLL zu versorgen. Zusätzlich muß h die Spaltennummer des linken Randes des zu verschiebenden Bereiches enthalten, l die oberste Zeile, d die rechte Spalte und e die unterste Zeile des Bereiches.
Beachten Sie, daß Spalte und Zeile 0 die linke obere Ecke des Bildschirmes darstellt. Achten Sie auch unbedingt selbst darauf, daß die übergebenen Parameter tatsächlich einen Bereich innerhalb des Video-RAMs markieren.

0E4F	SCR CHAR POSITION
0E5A	MC WAIT FLYBACK
0E64	SCR NEXT LINE
0E69	SCR NEXT LINE
0E76	SCR FLOOD BOX
0E8B	SCR CHAR POSITION
0E8F	SCR CHAR POSITION
0E93	MC WAIT FLYBACK
0E96	SCR PREV LINE
0E9B	SCR PREV LINE
0EE1	SCR NEXT BYTE
0EE5	SCR NEXT BYTE

0EF9 *** SCR UNPACK

Zeichenmatrix vergrößern (für Mode 0/1).

0EF9 SCR GET MODE

0F2A *** SCR REPACK

Zeichenmatrix wieder auf Originalform stauchen.

0F2B	SCR CHAR POSITION
0F2E	SCR GET MODE
0F3C	SCR NEXT LINE
0F48	SCR NEXT BYTE
0F53	SCR NEXT LINE
0F82	SCR NEXT BYTE
0F8C	SCR NEXT LINE

0F93 ************************************ SCR HORIZONTAL

Horizontale Linie ziehen.

0F9B ** SCR VERTICAL

Vertikale Linie ziehen.

0FA5	(GRA Pen)
0FA9	(GRA Pen)
0FAE	(GRA Pen)
0FB1	(GRA Pen)
0FB8	Akku mit &FF laden
0FF3	(GRA Paper)
0FFF	(GRA Pen)
100A	SCR NEXT BYTE
101C	(GRA Pen)
1027	(GRA Paper)
102C	SCR WIRTE
1030	SCR PREV LINE
1049	SCR DOT POSITION

1052 ** Default Farben

```
1052    04 04 0A 13 0C 0B 14 15
105A    0D 06 1E 1F 07 12 19 04
1062    17 04 04 0A 13 0C 0B 14
106A    15 0D 06 1E 1F 07 12 19
1072    0A 07
```

2.5.5 Text Screen (TXT)

Dieses Pack ist, wie der Name schon sagt, für die Verwaltung von Texten verantwortlich. Dazu gehört auch die Organisation der Windows.

Zu der Handhabung des Cursors sind ein paar Worte zu sagen:

Die in den Cursor-Routinen verlangten oder gelieferten Koordinaten sind als logische Angeben zu verstehen, d.h. sie beziehen sich auf das laufende Fenster. Die Koordinate 1,1 ist dabei die linke obere Ecke des Fensters. Wollen Sie, z.B. mit TXT SET CURSOR, den Cursor außerhalb des Fensters positionieren, wird er automatisch auf die nächst mögliche Position innerhalb des Fensters gesetzt, falls der Cursor eingeschaltet ist oder nachfolgend ein Zeichen dargestellt werden soll.
Dadurch wird auch die laufende Position (die Sie mit TXT GET CURSOR zurückbekommen) geändert.
Ist der Cursor ausgeschaltet, wird die gewünschte neue Position zunächst akzeptiert, bis entweder ein Zeichen dargestellt oder der Cursor eingeschaltet wird.

1074 *************************************** TXT INITIALISE

Vollständige Initialisierung des Text-Packs.

1074	TXT RESET
107E	TXT Default Params setzen
1081	Reset Params (alle Fenster)

1084 *** TXT RESET

Rücksetzen des Text-Packs.

1084 Restore TXT Indirections
1087 Move (hl+3) nach ((hl+1)), cnt=(hl)
108D db 0F 15 Bytes
108E dw BDCD Zieladresse
1090 TXT DRAW/UNDRAW CURSOR
1093 TXT DRAW/UNDRAW CURSOR
1096 TXT WRITE CHAR
1099 TXT UNWRITE CHAR
109C TXT OUT ACTION

109F ****************************** Reset Params (alle Fenster)

10A1 Start Params Fenster 0
10A4 lfd. Cursor Position (Row, Col)
10AF (lfd. Bildschirmfenster)
10B3 (lfd. Bildschirmfenster)
10BB TXT STR SELECT
10BE TXT DRAW/UNDRAW CURSOR
10C1 TXT GET PAPER
10C4 (TXT lfd. Paper)
10C7 TXT GET PEN
10CA (TXT lfd. Pen)
10D6 TXT STR SELECT
10DA (TXT lfd. Pen)
10DD Default Params setzen

10E4 ************************************** TXT STR SELECT

Textfenster wählen.

10E6 lfd. Bildschirmfenster
10F1 Adr. Fenster Params nach de
10F4 ldir cnt=15
10F8 Adr. Fenster Params nach de
10FC ldir cnt=15

1103 *********************************** TXT SWAP STREAMS

Die Parameter (Farben, Fenstergrenzen usw.) zweier Fenster werden miteinander vertauscht.

1103 (lfd. Bildschirmfenster)
1108 TXT STR SELECT
110C (lfd. Bildschirmfenster)
110F Adr. Fenster Params nach de
1114 Adr. Fenster Params nach de
1118 ldir cnt=15
111C TXT STR SELECT

111E ** ldir cnt=15

1126 ****************************** Adr. Fenster Params nach de

1135 lfd Cursor Position (Row, Col)

1139 *********************************** Default Params setzen

113C (lfd. Cursor Flag)
1140 TXT SET PAPER
1144 TXT SET PEN
1148 TXT SET GRAPHIC
114B TXT SET BACK
1154 TXT WIN ENABLE
1157 TXT VDU ENABLE

115A ************************************ TXT SET COLUMN

Horizontale Position des Cursors setzen.

115B lfd. Fenster links
115F (lfd. Cursor Pos. (Row, Col))

1165 *** TXT SET ROW

Vertikale Position des Cursors setzen.

1166 lfd. Fenster oben
116A (lfd. Cursor Pos. (Row, Col)

1170 ************************************* TXT SET CURSOR

Cursor positionieren.

1170 lfd. Fenster oben, links + hl
1173 TXT DRAW/UNDRAW CURSOR
1176 (lfd. Cursor Pos. (Row, Col)
1179 TXT DRAW/UNDRAW CURSOR

117C ************************************ TXT GET CURSOR

Abfrage der momentanen Cursorposition.

117C (lfd. Cursor Pos. (Row, Col))
117F lfd. Fenster oben, links - hl
1182 (lfd. Roll Count)

1186 ****************************** lfd. Fenster oben, links + hl

1186 (lfd. Fenster oben)
118C (lfd. Fenster links)

1193 ****************************** lfd. Fenster oben, links - hl

1193 (lfd. Fenster oben)
119B (lfd. Fenster links)

11A4 *** Move Cursor

11A4 TXT DRAW/UNDRAW CURSOR
11A7 (lfd. Cursor Pos. (Row, Col))
11AA hl innerhalb Fenstergrenzen?

11AD (lfd. Cursor Pos. (Row, Col))
11B2 lfd. Roll Count
11BA TXT GET WINDOW
11BD (TXT lfd. Paper)
11C1 SCR SW ROLL
11C5 SCR HW ROLL

11CA ************************************** TXT VALIDATE

Cursor innerhalb des Textfensters?

11CA lfd. Fenster oben, links + hl
11CD hl innerhalb Fenstergrenzen?
11D1 lfd. Fenster oben, links - hl

11D6 ****************************** hl innerhalb Fenstergrenzen

11D6 (lfd. Fenster rechts)
11DD (lfd. Fenster links)
11E2 (lfd. Fenster links)
11E7 (lfd. Fenster rechts)
11EF (lfd. Fenster oben)
11F7 (lfd. Fenster unten)

1208 ************************************** TXT WIN ENABLE

Größe des lfd. Textfensters festlegen.

1208 SCR CHAR LIMITS
1229 (lfd. Fenster oben)
122C (lfd. Fenster unten)
123A (Fenst. Flag (0=ges. Bildsch.))

1252 ************************************ TXT GET WINDOW

Welche Größe hat das lfd. Textfenster?

1252 (lfd. Fenster oben)
1255 (lfd. Fenster unten)
1259 (Fenst. Flag (0=ges. Bildsch.))

125F ************************ TXT DRAW/UNDRAW CURSOR

Setzen/Löschen des Cursors.

125F (lfd. Cursor Flag)

1265 ************************ TXT PLACE/REMOVE CURSOR

Cursor auf den Bildschirm setzen/Cursor vom Bildschirm nehmen.

126B (TXT lfd. Pen)
126F SCR CHAR INVERT

1276 *** TXT CUR ON

Cursor erlauben (Betriebssystem).

1279 Cur Enable Cont'd

127E ** TXT CUR OFF

Cursor verriegeln (Betriebssystem, höhere Priorität als TXT CUR ENABLE und TXT CUR DISABLE.

1281 Cur Disable Cont'd

1286 ************************************ TXT CUR ENABLE

Cursor erlauben (Anwenderprogramm).

1288 ************************************** Cur Enable Cont'd

1289 TXT DRAW/UNDRAW CURSOR
128E lfd. Cursor Flag
1294 TXT DRAW/UNDRAW CURSOR

1297 ************************************* TXT CUR DISABLE

Cursor verriegeln (Anwenderprogramm).

1299 ************************************** Cur Disable Cont'd

129A TXT DRAW/UNDRAW CURSOR
129F lfd. Cursor Flag

12A6 ** TXT SET PEN

Vordergrundfarbe setzen.

12A6 TXT lfd. Pen

12AB ** TXT SET PAPER

Hintergrundfarbe setzen.

12AB TXT lfd Paper
12AF TXT DRAW/UNDRAW CURSOR
12B3 SCR INK ENCODE
12B7 TXT DRAW/UNDRAW CURSOR

12BA ** TXT GET PEN

Welche Vordergrundfarbe ist gesetzt?

12BA (TXT lfd. Pen)
12BD SCR INK DECODE

12C0 ** TXT GET PAPER

Welche Hintergrundfarbe ist gesetzt?

12C0 (TXT lfd. Paper)
12C3 SCR INK DECODE

12C6 *** TXT INVERSE

Aktuelle Vorder- und Hintergrundfarbe austauschen.

12C6 TXT DRAW/UNDRAW CURSOR
12C9 (TXT lfd. Pen)
12CF (TXT lfd. Pen)

12D4 ************************************** TXT GET MATRIX

Adresse des Punktmusters eines Zeichens holen.

12D6 TXT GET M TABLE

12F2 ************************************** TXT SET MATRIX

Adresse des (vom Anwender definierten) Punktmusters eines bestimmten Zeichens setzen.

12F3 TXT GET MATRIX

12FE ************************************** TXT SET M TABLE

Startadresse und erstes Zeichen einer vom Anwender definiereten Punktmatrix setzen.

130A TXT GET MATRIX
131E TXT GET M TABLE
1321 (1. Zeichen User Matrix)
1326 (Adr. User Matrix)

132B ************************************ TXT GET M TABLE

Startadresse und erstes Zeichen einer Anwendermatrix?

132B (1. Zeichen User Matrix)
1331 (Adr. User Matrix)

1335 *** TXT WR CHAR

Zeichen darstellen.

1336 (lfd. Cursor Flag)
133C move Cursor
1340 (lfd. Cursor Pos. (Row, Col))
1345 TXT WRITE CHAR
1348 TXT DRAW/UNDRAW CURSOR

134B ************************************* TXT WRITE CHAR

Ein Zeichen auf den Bildschirm schreiben.

134C TXT GET MATRIX
1353 SCR UNPACK
1358 SCR CHAR POSITION
1366 SCR NEXT BYTE
136F SCR NEXT LINE
1377 (lfd. Background Mode)

137B ** TXT SET BACK

Transparentmodus ein/aus.

1384 (lfd. Background Mode)

1388 ************************************** TXT GET BACK

Welcher Transparentmodus?

1388 (lfd. Background Mode)
1392 (TXT lfd. Pen)
13A0 (TXT lfd. Pen)
13A5 SCR PIXELS

13A8 *********************************** TXT SET GRAPHIC

Darstellung von Steuerzeichen ein- oder ausschalten.

13A8 (GRA Char WR Mode (0=disable))

13AC ************************************** TXT RD CHAR

Zeichen vom Bildschirm lesen.

13AF move Cursor
13B2 TXT UNWRITE CHAR
13B6 TXT DRAW/UNDRAW CURSOR

13BE ******************************** TXT UNWRITE CHAR

Ein Zeichen vom Bildschirm lesen.

13BE (TXT lfd. Pen)
13C6 SCR REPACK
13DE SCR REPACK
13E4 TXT GET MATRIX

13FE *************************************** TXT OUTPUT

(Steuer-)Zeichen darstellen oder ausführen.

Bringt das Zeichen im Akku auf das lfd. Bildschirmfenster, bzw. führt es aus, falls es sich um ein Steuerzeichen handelt.

Beachten Sie, daß diese Routine die Indirection TXT OUT ACTION benutzt. Sollten Sie diese 'verbogen' haben, wird TXT OUTPUT auch Ihre Routine benutzen und nicht die ROM-Routine.

1402 TXT OUT ACTION

140A ************************************ TXT OUT ACTION

Ausgabe eines Zeichens auf dem Bildschirm oder Ausführung eines Steuercodes.

140B	(GRA Char WR Mode (0=disable)
1410	GRA WR CHAR
1413	Zeichenzähler Control Buffer
1418	Control Buffer voll?
141A	ja, dann springe
141C	Control Buffer leer?
141D	nein, dann springe
1420	Steuerzeichen?
1422	nein, dann TXT WR CHAR
1425	Zähler+1
142C	(Start Control Buffer)
1430	Sprungtabelle Steuerzeichen
1436	erforderliche Anzahl
1439	Steuerparameter erreicht
143A	nein, dann springe
1446	Start Control Buffer
144A	call (de)
144E	(Zeichenzähler Control Buffer)

1452 ************************************ TXT VDU DISABLE

Zeichendarstellung unterbinden.

1454 Cur Disable Cont'd

1459 ************************************ TXT VDU ENABLE

Es können Zeichen auf den Bildschirm geschrieben werden.

1445B Cur Enable Cont'd

1460 ******************* LFD. CURSOR FLAG NACH AKKU

1460 (lfd. Cursor Flag)

1464 ****************** Default Steuerzeichen Sprünge kopieren

1465 (Zeichenzähler Control Buffer)
1468 Default Steuerzeichen Sprünge
146B Sprungtabelle Steuerzeichen
146E Anzahl Bytes
1471 Kopiere

1474 **************************** Default Steuerzeichen Sprünge

1474 db 80
1475 dw 1513 00

1477 db 81
1478 dw 1335 01 TXT WR CHAR

147A db 80
147B dw 1297 02 TXT CUR DISABLE

147D db 80
147E dw 1286 03 TXT CUR ENABLE

1480 db 81
1481 dw 0AE9 04 SCR SET MODE

1483 db 81
1484 dw 1940 05 GRA WR CHAR

```
1486    db 00
1487    dw 1459  06 TXT VDU ENABLE

1489    db 80
148A    dw 14E1  07 Klingel

148C    db 80
148D    dw 1519  08 CRSR Left

148F    dw 80
1490    dw 151E  09 CRSR Right

1492    db 80
1493    dw 1523  0A CRSR Down

1495    db 80
1496    dw 1528  0B CRSR Up

1498    db 80
1499    dw 154F  0C TXT CLEAR WINDOW

149B    db 80
149C    dw 153F  0D CRSR auf Zeilenanfang

149E    db 81
149F    dw 12AB  0E TXT SET PAPER

14A1    db 81
14A2    dw 12A6  0F TXT SET PEN

14A4    db 80
14A5    dw 155E  10 Zeichen auf CRSR-Pos. löschen

14A7    db 80
14A8    dw 1599  11 Zeile bis CRSR-Pos. löschen

14AA    db 80
14AB    dw 158F  12 Zeile ab CRSR-Pos. löschen
```

```
14AD    db 80
14AE    dw 1578  13 Fenster bis CRSR-Pos. löschen

14B0    db 80
14B1    dw 1565  14 Fenster ab CRSR-Pos. löschen

14B3    db 80
14B4    dw 1452  15 TXT VDU DISABLE

14B6    db 81
14B7    dw 14EC  16 Transparentmode Ein/Aus

14B9    db 81
14BA    dw 0C55  17 SCR ACCESS

14BC    db 80
14BD    dw 12C6  18 TXT INVERSE

14BF    db 89
14C0    dw 150D  19 SYMBOL-Befehl

14C2    db 84
14C3    dw 1501  1A Fenster definieren

14C5    db 00
14C6    dw 14EB  1B kein Effekt

14C8    db 83
14C9    dw 14F1  1C INK-Befehl

14CB    db 82
14CC    dw 14FA  1D BORDER-Befehl

14CE    db 80
14CF    dw 1539  1E CRSR Home

14D1    db 82
14D2    dw 1547  1F LOCATE-Befehl
```

14D4 ********************************** TXT GET CONTROLS

Adresse der Steuerzeichen-Sprungtabelle holen.

14D4 Sprungtabelle Steuerzeichen

14E1 ** Klingel

14E6 SOUND QUEUE

14EC ********************************** Transparentmode Ein/Aus

14EE TXT SET BACK

14F1 *** INK-Befehl

14F7 SCR SET INK

14FA *** BORDER-Befehl

14FE SCR SET BORDER

1501 ** Fenster definieren

150A TXT WIN ENABLE

150D ** SYMBOL-Befehl

1510 TXT SET MATRIX

1513 Move Cursor
1516 TXT DRAW/UNDRAW CURSOR

1519 ** CRSR Left

151E *** CRSR Right

1523 ** CRSR Down

1528 ** CRSR Up

152C Move Cursor

1539 ** CRSR Home

153F ************************************ CRSR auf Zeilenanfang

153F Move Cursor
1542 (lfd. Fenster links)

1547 *** LOCATE-Befehl

154C TXT SET CURSOR

154F ************************************ TXT CLEAR WINDOW

Lfd. Textfenster löschen.

154F TXT DRAW/UNDRAW CURSOR
1552 (lfd. Fenster oben)
1555 (lfd. Cursor Pos. (Row, Col)
1558 (lfd. Fenster unten)

155E **************************** Zeichen auf CRSR-Pos. löschen

155E Move Cursor

1565 ****************************** Fenster ab CRSR-Pos. löschen

1565 12 Zeilen ab CRSR-Position löschen
1568 (lfd. Fenster oben)
156B (lfd. Fenster unten)
156F (lfd. Cursor Pos. (Row, Col))

1578 *************************** Fenster bis CRSR-Pos. löschen

1578 11 Zeilen bis CRSR-Position löschen
157B (lfd. Fenster oben)
157E (lfd. Fenster rechts)
1582 (lfd. Cursor Pos. (Row, Col)
1589 (TXT lfd. Paper)
158C SCR FILL BOX

158F ****************************** Zeile ab CRSR-Pos. löschen

158F Move Cursor
1593 (lfd. Fenster rechts)

1599 ***************************** Zeile bis CRSR-Pos. löschen

1599 Move Cursor
159E (lfd. Fenster links)
15A5 TXT DRAW/UNDRAW CURSOR

2.5.6 Graphics Screen (GRA)

Dieses Pack dient ausschließlich der Handhabung des Graphikfensters.

Zu den Koordinatenangaben, die von den verschiedenen Routinen verlangt werden, ist folgendes zu bemerken:

Die Koordinaten werden in drei Stufen übersetzt. Die anwendernächste Stufe ist die Position bezüglich des von ihm gesetzten Koordinatenursprungs (ORIGIN). Diese wird umgerechnet in eine Position relativ zum Bildschirmursprung (unten links). Diese beiden Stufen sind vom Mode unabhängig!
Die letzte Stufe ist die physikalische Adresse des Punktes. Diese ist abhängig vom lfd. Modus!

Diesen drei Stufen wird dann noch eine vierte Stufe vorangestellt, wenn ein relatives Koordinatenpaar in eine absolute Position, relativ zu ORIGIN, umgerechnet werden muß.

15A8 *************************************** GRA INITIALISE

Vollständige Initialisierung des Graphik-Packs.

15A8 GRA RESET
15AB Pen 1,Paper 0
15AF GRA SET PAPER
15B3 GRA SET PAPER
15B6 Origin auf 0,0 setzen
15BB GRA SET ORIGIN
15C6 GRA WIN WIDTH
15CB GRA WIN HEIGHT
15CE GRA GET PAPER
15D2 GRA GET PEN

15D7 *** GRA RESET

Zurücksetzen des Graphik-Packs.

15DA Restore GRA Indirections
15DD Move (hl+3) nach ((hl+1)), cnt=(hl)
15E0 db 09 9 Bytes
15E1 dw BDDC Zieladresse
15E3 GRA PLOT
15E6 GRA TEST
15E9 GRA LINE

15EC *** NN

15ED SCR ACCESS
15F1 GRA FILL

15FB ********************************* GRA MOVE RELATIVE

Bewegung relativ zur momentanen Position.

15FB Add lfd Koord. + rel Koord.

15FE ********************************* GRA MOVE ABSOLUTE

Bewegung zu einer absoluten Position.

15FE (lfd X Koord.)
1602 (lfd Y Koord.)

1606 ************************************** GRA ASK CURSOR

Wo ist der lfd. Graphikcursor?

1606 (lfd X Koord.)
160A (lfd Y Koord.)

160E ************************************** GRA SET ORIGIN

Ursprung der Anwender-Koordinaten setzen.

160E (X Origin)
1612 (Y Origin)
161A GRA MOVE ABSOLUTE

161C ************************************** GRA GET ORIGIN

Ursprung der Anwender-Koordinaten holen.

161C (X Origin)
1620 (Y Origin)

1624 ************************************ phys Startposition holen

1624 GRA ASK CURSOR

1627 ******************** phys Zielposition holen + Cur. setzen

1627 GRA MOVE ABSOLUTE

162A ********************** GRA KOORD. KONVERTIEREN

162B SCR GET MODE
1640 (X Origin)
1655 (Y Origin)

165D ***************************** Add lfd Koord. + rel Koord.

165E (lfd X Koord.)
1664 (lfd Y Koord.)
166A (X Koord. GRA Fenster links)
1673 (X Koord. GRA Fenster rechts)
1680 (Y Koord. GRA Fenster oben)
1689 (Y Koord. GRA Fenster unten)
1694 phys Zielposition holen + Cur. setzen

16A5 *************************************** GRA WIN WIDTH

Linke und rechte Begrenzung des Graphikfensters setzen.

16BE SCR Get Mode
16C9 (X Koord. GRA Fenster links)
16CD (X Koord. GRA Fenster rechts)

16EA ************************************* GRA WIN HEIGHT

Obere und untere Begrenzung des Graphikfensters setzen.

16FB (Y Koord. GRA Fenster oben)
16FF (Y Koord. GRA Fenster unten)

1717 ************************************ GRA GET W WIDTH

Linke und rechte Begrenzung des Graphikfensters?

1717 (X Koord. GRA Fenster links)
171B (X Koord. GRA Fenster rechts)
171E SCR GET MODE

172D *********************************** GRA GET W HEIGHT

Obere und untere Begrenzung des Graphikfensters?

172D (Y Koord. GRA Fenster oben)
1731 (Y Koord. GRA Fenster unten)

1736 *********************************** GRA CLEAR WINDOW

Graphikfenster löschen.

1736 GRA GET W WIDTH
1746 (Y Koord. GRA Fenster unten)
174A (Y Koord. GRA Fenster oben)
1753 (X Koord. GRA Fenster links)
1759 SCR DOT POSITION

175D (GRA Paper)
1761 SCR FLOOD BOX

1767 ** GRA SET PEN

Schreibfarbe setzen.

1767 SCR INK ENCODE
176A (GRA Pen)

176E *************************************** GRA SET PAPER

Hintergrundfarbe setzen.

176E SCR INK ENCODE
1771 (GRA Paper)

1775 ** GRA GET PEN

Welche Schreibfarbe?

1775 (GRA Pen)

177A ************************************** GRA GET PAPER

Welche Hintergrundfarbe?

177A (GRA Paper)
177D (SCR INK DECODE)

1780 ********************************** GRA PLOT RELATIVE

Graphikpunkt relativ zur aktuellen Cursorposition setzen.

1780 Add lfd. Koord. + rel. Koord.

1783 ********************************* GRA PLOT ABSOLUTE

Graphikpunkt setzen (absolut).

1783 GRA PLOT

1786 *** GRA PLOT

Stellen einen Punkt auf dem Bildschirm dar.

178A SCR DOR POSITION
178D (GRA Pen)
1791 SCR WRITE

1794 ********************************* GRA TEST RELATIVE

Punkt gesetzt (relativ zum lfd. Cursor)?

1794 Add. lfd. Koord. + rel. Koord.

1797 ********************************* GRA TEST ABSOLUTE

Punkt gesetzt (absolut)?

1797 GRA TEST

179A *** GRA TEST

Gib die Ink der momentanen Graphik-Position.

179D GRA GET PAPER
17A0 SCR DOT POSITION
17A3 SCR READ

17A6 ********************************* GRA LINE RELATIVE

Linie von der lfd. bis zur relativen Distanz ziehen.

17A6 Add. lfd. Koord. + rel. Koord.

17A9 ********************************* GRA LINE ABSOLUTE

Linie von der lfd. zur absoluten Position ziehen.

17A9 GRA LINE

17AC ************************** GRA MASK PARAM RETTEN

Parameter aus dem BASIC-Befehl MASK retten.

17B0 ************************** GRA MASK PARAM RETTEN

Parameter aus dem BASIC-Befehl MASK retten.

17B4 *** GRA LINE

Zeichne eine Linie.

17B9	phys. Zielposition holen + Cur. setzen
17BD	(Rechenpuffer X Koord.)
17CC	(Rechenpuffer Y Koord.)
188C	phys. Startposition holen
188F	(Rechenpuffer X Koord.)
1893	(Rechenpuffer Y Koord.)
18A2	(Rechenpuffer Y Koord.)
18AD	(Rechenpuffer Y Koord.)
18B2	(Rechenpuffer Y Koord.)
18B9	(Y Koord. GRA Fenster oben)
18C3	(Y Koord. GRA Fenster unten)
18C8	(Rechenpuffer X Koord.)
18DA	(Rechenpuffer X Koord.)
18E6	(Rechenpuffer X Koord.)
18EF	(Rechenpuffer X Koord.)
18FA	(Rechenpuffer X Koord.)
18FF	(Rechenpuffer X Koord.)
1906	(X Koord. GRA Fenster rechts)
1910	(X Koord. GRA Fenster links)

1915 (Rechenpuffer Y Koord.)
1928 (Rechenpuffer Y Koord.)
1934 (Rechenpuffer Y Koord.)

1940 *** GRA WR CHAR

Ein Zeichen an der lfd. Graphikcursor-Position schreiben.

1942 TXT GET MATRIX
1948 phys. Startposition holen
1962 SCR DOR POSITION
1973 SCR NEXT BYTE
197B SCR NEXT LINE
1985 GRA ASK CURSOR
1989 SCR GET MODE
1998 GRA MOVE ABSOLUTE
19AC SCR DOT POSITION
19C4 (GRA Pen)
19CE (GRA Paper)
19D2 SCR WRITE

19D5 ********************************** GRA PARAM RETTEN

19D9 ** GRA FILL

19D9 (Rechenpuffer X Koord.)
19DF (Rechenpuffer Y Koord.)
19E3 SCR INK ENCODE
19E9 phys. Startposition holen
1A19 (Rechenpuffer X Koord.)
1A25 (Rechenpuffer Y Koord.)
1A2C (Rechenpuffer Y Koord.)
1A44 (Rechenpuffer X Koord.)
1A9F (Rechenpuffer Y Koord.)
1AA9 (Rechenpuffer Y Koord.)
1AC1 (Rechenpuffer X Koord.)

1AE8 (GRA Y Koord. GRA Fenster oben)
1B10 SCR PREV LINE
1B18 (Y Koord. GRA Fenster unten)
1B25 SCR NEXT LINE
1B35 (GRA Pen)
1B45 SCR DOT POSITION
1B51 SCR DOT POSITION
1B56 SCR DOT POSITION

2.5.7 Keyboard Manager (KM)

Diesem Pack obliegt die Überwachung der Tastatur und die Umsetzung in brauchbare Zeichencodes. Bei der Erfüllung dieser Aufgabe, also der zyklischen Abfrage der Tasten, bedient es sich des EVENT-Mechanismus.

1B5C *** KM INITIALISE

Vollständige Initialisierung der Tastaturverwaltung. Der Zustand der Tastaturverwaltung vor dem Aufruf von KM INITIALISE geht verloren.

1B5F	KM SET DELAY
1B68	(Shift Lock State)
1B80	Key Translation Table
1B8A	Key State Map
1B8D	während Scan gedrückte Keys

1B98 *** KM RESET

Die Tastaturverwaltung wird in ihren Ausgangszustand gebracht. Die indirekte Sprungtabelle und die Puffer der Tastaturverwaltung werden neutralisiert.

1BA4	Exp Buffer Cont'd
1BA7	Restore KM Indirection
1BAA	Move (hl+3) nach ((hl+1)), cnt=(hl)
1BB0	KM DISARM BREAK
1BB3	db 03 3 Bytes
1BB4	dw BDEE Zieladresse
1BB6	Test Break

1BBF ************************************** KM WAIT CHAR

Holt ein Zeichen aus dem Eingabepuffer, bzw. Expansion-String oder Put Back Buffer. Falls kein Zeichen verfügbar ist, kehrt die Routine nicht zurück, sondern wartet. Im Erfolgsfalle enthält der Akku das eingegebene Zeichen.

1BBF KM READ CHAR
1BC2 KM WAIT CHAR

1BC5 ************************************** KM READ CHAR

Holt ebenfalls ein Zeichen (vgl. KM WAIT CHAR), wenn eines vorhanden ist. Wartet nicht auf das nächste Zeichen. Falls nach Rückkehr aus der Routine das carry gesetzt ist, war der Versuch erfolglos.

1BC6 Put Back Buffer
1BC9 Zeichen holen
1BCA Puffer löschen
1BCC war ein Zeichen da?
1BCD wenn ja springe
1BCF (Exp. String Pointer)
1BD2 Highbyte nach Akku
1BD3 Exp. String vorhanden?
1BD4 wenn ja springe
1BD6 KM READ KEY
1BD9 springe wenn kein Zeichen
1BDB ist das Zeichen < 128?
1BDD wenn < 128 dann springe
1BE8 KM GET EXPAND
1BF0 (Exp. String Pointer)

1BF8 Akku=&FF

1BFA ************************************ KM CHAR RETURN

Ein Zeichen im Tastaturpuffer für den nächsten Zugriff (KM READ CHAR oder KM WAIT CHAR) hinterlegen.

1BFA (Put Back Buffer)

1BFE KM READ CHAR

1C04 ************************************** KM EXP BUFFER

Speicher für Erweiterungsstring zuweisen (Adresse, Länge). Puffer initialisieren.

1C04 Exp Buffer Cont'd

1C0A ************************************** Exp Buffer Cont'd

1C13 (Pointer Ende Exp Buffer)
1C17 (Pointer Start Exp Buffer)
1C1A ASCII
1C1D 0
1C1F bis
1C20 9
1C21 nach
1C22 Expansion
1C23 Buffer
1C25 Restore
1C26 Default Exp String
1C35 (Pointer freier Exp Buffer)

1C3C *************************************** Default Exp String

```
1C3C    01 2E 01 0D 05 52 55 4E   .....RUN
1C44    22 0D                     ".
```

1C46 ************************************** KM SET EXPAND

Erweiterungsstring einrichten.

1C47 Adresse Exp String nach de
1C4A springe wenn Token ungültig
1C4E Exp Buffer aufräumen

1C6A *********************************** Exp Buffer aufräumen

1C79 Platz für neuen Exp String?
1C85 (Pointer freier Exp Buffer)
1C8A (Pointer Ende Exp Buffer)
1C93 Platz für neuen Exp String?
1C96 (Pointer freier Exp Buffer)
1CA1 (Pointer freier Exp Buffer)

1CA7 ****************************** Platz für neuen Exp String?

1CA7 (Pointer freier Exp Buffer)

1CB3 ************************************ KM GET EXPAND

Zeichen vom Erweiterungsstring holen. Beginnend bei Null, sind die Zeichen der Zeichenkette fortlaufend durchnumeriert.

1CB3 Adresse Exp String nach de

1CC3 ****************************** Adresse Exp String nach de

1CC3 Token im gültigen
1CC5 Bereich?
1CC7 Rücksprung wenn nicht
1CC9 (Pointer Start Exp Buffer)
1CD0 hl um die Länge des
1CD1 Expansion String
1CD2 weitersetzen

1CDB ** KM WAIT KEY

Wartet auf den nächsten Tastendruck, wenn nicht unmittelbar ein Zeichen verfügbar ist. Untersucht nur den Eingabepuffer, nicht aber Expansion-String und Put Back Buffer (vgl. KM WAIT CHAR).

1CDB KM READ KEY
1CDE KM WAIT KEY

1CE1 ** KM READ KEY

Holen der Tastennummer, falls eine Taste gedrückt wurde. Wartet nicht, wenn nicht unmittelbar ein Zeichen zur Verfügung steht. Expansion-String und Put Back Buffer bleiben unberücksichtigt.

1CFB Caps Lock State
1D12 Shift Lock State
1D17 caps lock?
1D1A wenn nicht springe
1D1D toggle caps lock
1D27 KM GET CONTROLS
1D2B (Shift Lock State)
1D32 KM GET SHIFT
1D35 KM GET TRANSLATE

1D38 ** KM GET STATE

Untersuche ob CAPS-LOCK- und SHIFT-LOCK-Taste betätigt wurden.

1D38 (Shift Lock State)

1D3C ** Set State

1D3C (Shift Lock State)

1D40 ************************ KM UPDATE KEY STATE MAP

1D40 Mulitihit Kontr. zu B63F
1D43 während Scan gedrückte Keys
1D46 Scan Keyboard
1D4C SHIFT/CTRL isolieren
1D4F Key 16...23
1D54 Mulitihit Kontr. zu B63F
1D57 Key State Map
1D74 Test Break
1D86 Key State Map
1D8B (Adresse der Repeat Tabelle)
1D9E (KM Delay)

1DB8 ************************************** KM TEST BREAK

1DC1 KM BREAK EVENT
1DCE KM BREAK EVENT

1DE5 ********************************** KM GET JOYSTICK

Der Zustand der Joysticks zum Zeitpunkt der Abfrage wird mit Hilfe der Key State Map ermittelt.

1DE5 (Joystick 1)
1DEB (Joystick 0)

1DF2 ************************************** KM GET DELAY

Parameter für Tastenwiederholungseinsatz und -geschwindigkeit holen.

1DF2 (KM Delay)

1DF6 ************************************** KM SET DELAY

Tastenwiederholgungseinsatz und -geschwindigkeit setzen.

1DF6 (Km Delay)

1DFA ************************************* KM ARM BREAK

Zulassen der Break-Taste.

1DFA   KM DISARM BREAK
1DFD   Break Event Block
1E02   KL INIT EVENT

1E0B *********************************** KM DISARM BREAK

Die Break-Taste wird verriegelt.

1E13   KL DEL SYNCHRONOUS

1E19 ************************************ KM BREAK EVENT

Routinen beim Betätigen der Break-Taste ausführen.

1E24   KL EVENT

1E2F ************************************** KM GET REPEAT

Überprüfung, ob es sich bei einer bestimmten Taste um eine Taste mit gesetzter Wiederholungsfunktion handelt.

1E2F   (Adresse der Repeat Tabelle)
1E32   Z entspr. Key Bit setzen

1E34 *************************************** KM SET REPEAT

Durch einen Eintrag in die Wiederholungstastentabelle wird festgelegt, ob die Taste Wiederholungsfunktion hat. Im Akku ist dabei die Tastennummer zu hinterlegen. Soll die Taste repetieren, so muß b &FF enthalten. Enthält b hingegen &00, so wird die Wiederholfunktion für die betreffende Taste aufgehoben.

1E34   Key > 80?
1E36   ja dann ungültig

1E37 (Adresse der Repeat Tabelle)
1E3A der Key# entspr. Bit holen

1E45 *** KM TEST KEY

Mit dem Zustand der Key State Map wird untersucht ob eine Taste oder ein Joystick betätigt wurde.

1E46 (Key 16...23)
1E49 SHIFT/CTRL isolieren
1E4D Key State Map
1E50 der Key# entspr. Bit holen
1E53 Key Bit maskieren

1E55 ****************************** der Key# entspr. Bit holen

1E57 Key#
1E59 /8
1E5F Key Map adressieren
1E62 Bit Masken
1E65 dem Key
1E67 entsprechendes
1E68 Bit
1E69 laden

1E6D *** Bit Masken

1E6D 01 02 04 08 10 20 40 80

1EC4 ********************************* KM GET TRANSLATE

Eintrag aus der ersten Ebene der Tastaturtabelle (Key State Map) holen.

1EC4 (Adresse Key Transl. Table)
1EC7 Get Key Table

1EC9 *** KM GET SHIFT

Eintrag aus der zweiten Ebene der Tastaturtabelle holen.

1EC9 (Adresse Key SHIFT Table)
1ECC Get Key Table

1ECE ************************************** KM GET CONTROL

Eintrag aus der dritten Ebene der Tastaturtabelle holen.

1ECE (Adresse Key CTRL Table)

1ED1 *** Get Key Table

1ED8 ********************************** KM SET TRANSLATE

Eintrag in die erste Ebene der Tastaturtabelle vornehmen.

1ED8 (Adresse Key Transl. Table)
1EDB Set Key Table

1EDD *** KM SET SHIFT

Eintrag in die zweite Ebene der Tastaturtabelle vornehmen.

1EDD (Adresse Key SHIFT Table)
1EE0 Set Key Table

1EE2 ************************************** KM SET CONTROL

Eintrag in die dritte Ebene der Tastaturtabelle vornehmen.

1EE2 (Adresse Key CTRL Table)

1EE5 *** Set Key Table

1EEF ************************************ Key Translation Table

```
1EEF   F0 F3 F1 89 86 83 8B 8A
1EF7   F2 E0 87 88 85 81 82 80
1EFF   10 5B 0D 5D 84 FF 5C FF
1F07   5E 2D 40 70 3B 3A 2F 2E
1F0F   30 39 6F 69 6C 6B 6D 2C
1F17   38 37 75 79 68 6A 6E 20
1F1F   36 35 72 74 67 66 62 76
1F27   34 33 65 77 73 64 63 78
1F2F   31 32 FC 71 09 61 FD 7A
1F37   0B 0A 08 09 58 5A FF 7F
```

1F3F *** Key SHIFT Table

```
1F3F   F4 F7 F5 89 86 83 8B 8A
1F47   F6 E0 87 88 85 81 82 80
1F4F   10 7B 0D 7D 84 FF 60 FF
1F57   A3 3D 7C 50 2B 2A 3F 3E
1F5F   5F 29 4F 49 4C 4B 4D 3C
1F67   28 27 55 59 48 4A 4E 20
1F6F   26 25 52 54 47 46 42 56
1F77   24 23 45 57 53 44 43 58
1F7F   21 22 FC 51 09 41 FD 5A
1F87   0B 0A 08 09 58 5A FF 7F
```

1F8F *** Key CTRL Table

```
1F8F   F8 FB F9 89 86 83 8C 8A
1F97   FA E0 87 88 85 81 82 80
1F9F   10 1B 0D 1D 84 FF 1C FF
1FA7   1E FF 00 10 FF FF FF FF
1FAF   1F FF 0F 09 0C 0B 0D FF
1FB7   FF FF 15 19 08 0A 0E FF
1FBF   FF FF 12 14 07 06 02 16
1FC7   FF FF 05 17 13 04 03 18
1FCF   FF 7E FC 11 E1 01 FE 1A
1FD7   FF FF FF FF FF FF FF 7F
1FDF   07 03 4B FF FF FF FF FF
1FE7   AB 8F
```

2.5.8 Sound Manager (SOUND)

Über dieses Pack, obgleich es recht umfangreich ist, läßt sich nicht viel berichten. Die eigendliche Tonerzeugung nimmt nur einen verschwindend geringen Raum ein; hingegen der Löwenanteil von der Verwaltung verschiedener Warteschlangen eingenommen wird. Dazu zählt auch die Realisierung der **TONE ENVELOPE**, die der programmierbare Sound Generator (PSG) nicht von sich aus beherrscht.

Wenn Sie Ihrem CPC so richtig die Flötentöne beibringen wollen, so empfiehlt es sich den PSG unmittelbar zu programmieren, denn die Routinen des **SOUND MANAGERs** sind allzu sehr auf die zugehörigen BASIC-Befehle abgestimmt. Unter BASIC kann der CPC zwar ein flottes Liedchen trällern, doch um ein fetziges Schlagzeug zu programmieren, muß man schon in die Maschinensprache einsteigen. Erst dann wird es möglich komplexe Klangstrukturen in schneller Abfolge zu verschachteln.

1FE9 ** SOUND RESET

Rücksetzen des gesamten SOUND MANAGERs. Löschen aller Warteschlangen.

1FF3 Sound Event
1FF8 KL INIT EVENT
2000 SOUND Params Kanal A

2050 *** SOUND HOLD

Anhalten aller Töne, kann durch SOUND CONTINUE rückgängig gemacht werden.

2050 lfd. SOUND Aktivität

2058 Kanäle aktiv?
2059 wenn nicht Rücksprung
205C Lautstärke
205E aller Kanäle
2060 auf 0
2063 MC SOUND REGISTER

206B ************************************ SOUND CONTINUE

Zuvor angehaltene Töne (SOUND HOLD) weiter bearbeiten.

206B (alte SOUND
206E Akt. (nach HOLD))
206F Kanal aktiv?
2070 wenn nicht Rücksprung
2076 bei allen
2079 Kanälen
207A wieder alte
207D Lautstärke setzen

208B *** Sound Event

209D Kanal aktiv?
209F nein dann nächster

20D7 ************************************** Scan Sound Queues

20D7 lfd SOUND Aktivität
2111 KL EVENT

2114 *** SOUND QUEUE

Ton an die Warteschlange anhängen.

2114 SOUND CONTINUE

21AC ************************************* SOUND RELEASE

Töne erlauben.

21AD SOUND CONTINUE

21CE ************************************** SOUND CHECK

Ist noch Platz in der Warteschlange?

21EB ********************************** SOUND ARM EVENT

Eventblock für den Fall 'scharf machen', daß in der Warteschlange ein Platz frei wird.

2206	KL EVENT
2258	lfd SOUND Aktivität
227D	KL EVENT
2296	SOUND Params Kanal A
229E	SOUND Params Kanal B
22A6	SOUND Params Kanal C
22B8	SOUND Params Kanal C
22C0	SOUND Params Kanal B
22F3	Rauschgenerator laden
22F5	MC SOUND REGISTER
2303	Lautstärke Hüllkurven
2342	Lautstärke setzen
237D	Hüllkurve
237F	MC SOUND REGISTER
2383	Hüllkurvenlänge Lo
2385	MC SOUND REGISTER
2389	Hüllkurvenlänge Hi
238B	MC SOUND REGISTER
2390	Lautstärke setzen

23DB ************************************* Lautstärke setzen

23E2 Lautstärke
23E4 MC SOUND REGISTER
23EF lfd SOUND Aktivität
2403 Kanal-Steuerregister
2405 MC SOUND REGISTER
240C SOUND T ADRESS
2486 Tonhöhe Lo
2489 MC SOUND REGISTER
248F Tonhöhe Hi
2492 MC SOUND REGISTER

2495 **************************** SOUND AMPL ENVELOPE

Lautstärkehüllkurve einrichten (15 verschiedene Amplituden).

2495 Lautstärke Hüllkurven
2498 Hüllkurve kopieren

249A **************************** SOUND TONE ENVELOPE

Tonehüllkurve einrichten (15 verschiedene Ton-Hülkurven).

249A Ton Hüllkurven

249D ************************************ Hüllkurve kopieren

249E Hüllkurve Adresse holen

24A6 ************************************ SOUND A ADRESS

Adresse einer Lautstärkehüllkurve holen.

24A6 Lautstärke Hüllkurven
24A9 Hüllkurve Adresse holen

24AB ************************************ SOUND T ADRESS

Adresse einer Ton-Hüllkurve holen.

24AB Ton Hüllkurven

24AE ******************************* Hüllkurve Adresse holen

2.5.9 Cassette Manager (CAS)

Seien Sie unbesorgt, es ist uns nicht entgangen, daß Ihr Rechner über ein eingebautes Diskettenlaufwerk verfügt, was für Sie die Verwendung von Kassetten nahezu oder gar vollständig überflüssig werden läßt. Trotzdem wollen wir es nicht versäumen Ihnen auch den **CASSETTEN MANAGER** vorzustellen, der einige Basisroutinen enthält, die man kennen sollte.

24BC *** CAS INITIALISE

Vollständige Initialisierung des Kassetten-Packs

24BC CAS IN ABANDON
24C3 CAS NOISY

24CE *** CAS SET SPEED

Schreibgeschwindigkeit setzen.

24D9 (Cass. Speed)

24E1 ** CAS NOISY

Kassettenmeldungen ein/aus. Von einem Sperren der Kasettenmeldungen werden Fehlermeldungen ausgeschlossen.

24E1 (Cass. Message Flag)

24E5 ** CAS IN OPEN

Eröffnet ein Eingabefile. Dazu müssen in b die Länge des Filenamens, in hl die Anfangsadresse des Filenamens und in de die

Startadresse eines 2K großen RAM-Bereiches, der als Eingabepuffer verwendet wird, an die Routine übergeben werden.

Nach der Rückkehr enthält hl die Anfangsadresse des Fileheaders; a, bc und de enthalten weitere dem Header entnommene Werte. Diese Werte können Sie aber auch selbst dem Header entnehmen, dessen Anfangsadresse Sie ja in hl haben.

Die Flags carry und zero geben Auskunft über den Erfolgt der Aktion:

Carry=1 und zero=0 zeigt an, daß alles geklappt hat.
Carry=0 und zero=0 sagt aus, daß bereits ein anderes File geöffnet ist.
Carry=0 und zero=1 besagt, die ESC-Taste wurde gedrückt.

24E5 Input Buffer Status
24E9 Cass. Open
24ED File Header lesen

24FE ** CAS OUT OPEN

Ein Ausgabefile wird eröffnet. Die Übergabeparameter und die Bedeutung der Flags ist die gleiche wie bei CAS IN OPEN, mit dem Unterschied, daß de nun selbstverständlich die Startadresse des Ausgabepuffers enthalten muß.

24FE Output Buffer Status

2502 ** Cass. Open

2550 ** CAS IN CLOSE

Korrektes Schließen der Eingabedatei.

2550 (Input Buffer Status)

2557 ************************************** CAS IN ABANDON

Lesen sofort abbrechen und Eingabedatei schließen (im Falle eines Fehlers).

2557 Input Buffer Status

257F *************************************** CAS OUT CLOSE

Korrektes Schließen der Ausgabedatei.

257F (Output Buffer Status)

2599 ************************************* CAS OUT ABANDON

Ausgabedatei sofort schließen und Ausgebegerät als "geschlossen" kennzeichnen. Noch nicht geschriebene Daten werden vernichtet.

2599 Output Buffer Status

25A0 *** CAS IN CHAR

Holt ein Zeichen aus dem Eingabepuffer und übergibt es im Akku. War es das letzte Zeichen aus dem Puffer, so wird automatisch ein neuer Block von der Kassette in den Puffer gelesen.

Carry=0 und zero=0 bedeutet, daß das Dateiende (EOF) erreicht ist oder daß das File nicht geöffnet war. Alle anderen Kombinationen wie bei CAS IN OPEN

25A5 Check Input Buffer Status
25B0 File Header lesen
25BC (Pointer Input Buffer)
25BF ld a,(hl)
25C1 (Pointer Input Buffer)

25C6 ** CAS OUT CHAR

Schreibt das Zeichen im Akku in den Ausgabepuffer. Ist der Puffer voll, so wird er automatisch auf Kassette weggeschrieben.

Die Bedeutung der Flags entspricht der bei CAS IN CHAR und CAS IN OPEN.

25CA Output Buffer Status
25CF Check Buffer Status
25EA (Pointer Output Buffer)
25EF (Pointer Output Buffer)

25F6 ********************************** Check Input Buffer Status

25F6 Input Buffer Status

25F9 ** Check Buffer Status

2603 *** CAS TEST EOF

Abfrage ob das Dateiende erreicht wurde.

2603 CAS IN CHAR

2607 ** CAS RETURN

Zuletzt gelesenes Zeichen zurück in den Puffer.

260F (Pointer Input Buffer)
2613 (Pointer Input Buffer)

2618 ** CAS IN DIRECT

Gesamte Eingabedatei in den Speicher übertragen, kein zeichenweises lesen.

261B (Check Input Buffer Status)

2631 File Header lesen
263C (Adr. Start Input Buffer)
2647 KL LDIR CONT'D
2650 KL LDDR CONT'D

2653 ************************************ CAS OUT DIRECT

Definierten Speicherbereich auf Kassette schreiben (nicht über den Puffer).

2656 Output Buffer Status
265B Check Buffer Status
266E (Adr. Start Output Buffer)
2685 (Adr. Start Output Buffer)

2692 ** CAS CATALOG

Ausgabe eines Kataloges der Kassette aus dem Bildschirm.

2692 Input Buffer Status
269C (Adr. Start Input Buffer)
26A1 CAS NOISY
26A9 CAS IN ABANDON

26AC *************************************** File Header lesen

26C3 CAS READ
26E0 (Input Buffer Status)
26EF (Adr. Start Input Buffer)
26F2 (Pointer Input Buffer)
26F7 CAS READ
271B Input Buffer Status
2743 (File Header Input)
274E File Header Input
2760 File Header Input
277B CAS OUT CLOSE
2781 CAS MOTOR STOP
2790 File Header Output
279E (Adr. Start Output Buffer)

27A1 (Pointer Output Buffer)
27A8 File Header Output
27B0 CAS WRITE
27BC CAS WRITE
27D9 Output Buffer Status
27F5 CAS START MOTOR
2807 (Cass. Message Flag)
2846 TXT WR CHAR
2871 CAS Meldung (1 Zeichen) ausgeben
2886 CAS Meldung (# in b) ausgeben
288C CAS Meldung (1 Zeichen) ausgeben

2891 ************************ Cass. Meldung (# in b) ausgeben

2891 TXT GET CURSOR
289D Kassetten-Melungen
28C9 Cass. Meldung (1 Zeichen) ausgeben
28D0 Cass. Meldung (1 Zeichen) ausgeben
28D2 (Cass. Message Flag)
28D8 Cass. Meldung (# in b) ausgeben
28DB KM READ CHAR
28DE TXT CUR ON
28E1 KM WAIT KEY
28E4 TXT CUR OFF

28F0 ********************* Cass. Meldung (1 Zeichen) ausgeben

28F0 TXT OUTPUT
28F7 TXT SET COLUMN
28FE TXT GET WINDOW
2902 TXT GET CURSOR
2924 Cass. Meldung (1 Zeichen) ausgeben
292F (Input Buffer Status)

2935 *********************************** Kassetten-Meldungen

2935 Press
293B PLAY
293F then

2943 any
2946 key
294B error
2955 REC
2958 and
295D Read
2963 Write
296A Rewind
2970 tape
2975 Found
297D Loading
2985 Saving
298D ok
2990 block
2996 Unnamend
299D file

29A6 ** CAS READ

Eine Block von Kassette lesen. Diese Routine wird von übergeordneten Routinen angesprochen.

29A6 Motor Ein & Keyb. öffnen

29AF ** CAS WRITE

Einen Block auf Kassette schreiben. Wird wie CAS READ von übergeordneten Routinen aufgerufen.

29AF Motor Ein & Keyb. öffnen

29C1 ** CAS CHECK

Block auf dem Band mit Speicherinhalt vergleichen.

29C1 Motor Ein & Keyb. öffnen
29D2 Port A=Out
29D7 Motor ein
29DE CAS RESTORE MOTOR

29E3 ****************************** Motor Ein & Keyb. öffnen

29EA	SOUND RESET
29F0	CAS START MOTOR
29F4	Sound I/O Port select
29F9	Strobe ein
29FE	Strobe aus
2A02	Port A=In
2A07	Keyb. Y9 (ESC) öffnen
2A0A	& Sound I/O auf Port A
2A3C	RAM LAM (IX)
2A67	RAM LAM (IX)
2A95	Cass. Input RD DATA & Test ESC
2A9D	Cass. Input RD DATA & Test ESC
2AB2	Cass. Input RD DATA & Test ESC

2B3D ********************** Cass. Input RD DATA & Test ESC

2B3D	Port A
2B3F	Keyb. X
2B41	ESC?
2B43	wenn ja Rücksprung
2B4D	Port B
2B55	Input RD DATA
2B8E	WR DATA Aus
2B90	Cass. Output WR DATA
2B9F	WR DATA Ein
2BA1	Cass. Output WR DATA

2BA7 ******************************** Cass. Output WR DATA

2BB1	Port Control
2BB3	WR DATA

2BBB ********************************** CAS START MOTOR

Kassettenmotor ein

2BBD CAS RESTORE MOTOR

2BBF ************************************ CAS STOP MOTOR

Kassettenmotor stoppen.

2BC1 ********************************* CAS RESTORE MOTOR

Stellt alten Zustand des Motors wieder her. Nach dem Einschalten des Motors wird das Erreichen der Soll-Drehzahl abgewartet.

2BC2 Port C
2BCE Motor Ein/Aus
2BE9 KM TEST KEY

2.5.10 Screen Editor (EDIT)

Der SCREEN EDITOR wird im Gegensatz zu den bisher behandelten Bestandteilen des L ROMs vom Betriebssystem überhaupt nicht benutzt. Es handelt sich demnach strenggenommen nicht um ein Pack in dem Sinne, wie wir es bisher verstanden haben. Der SCREEN EDITOR kann vielmehr in Zusammenhang mit den Arithmetik-Routinen gebracht werden, die ebenfalls ausschließlich vom BASIC angesprungen werden.

Es scheint uns nicht sinnvoll zu sein die Routinen des SCREEN EDITORs einzeln zu nutzen, bestenfalls den Editor als Ganzes. Hierzu müssen Sie hl mit der Anfangsadresse Ihres zu editierenden Textes versorgen. Dieser Text darf maximal 255 Zeichen lang sein, was auch der größten Länge einer BASIC-Zeile entspricht.

2C02 ** EDIT

2C12	EDIT Sprung ausführen
2C1A	EDIT SPRUNG ausführen
2C1D	Zeiger auf Eingabepuffer
2C1E	Zeichen im Puffer zählen
2C24	(Insert Flag)
2C2D	Zeichen von Keyboard

2C42 ********************************** EDIT Sprung ausführen

2C49	EDIT Sprungtabelle 1
2C4E	Zeichen im Puffer?
2C50	springe wenn ja
2C52	eine der Cursor Tasten?
2C54	springe wenn nicht

2C56 Cursor Taste und SHFT/CTRL?
2C58 springe wenn ja
2C5A EDIT Sprungtabelle 2

2C72 ************************************ EDIT Sprungtabelle 1

2C72 db 13 Anzahl Einträge

2C73 dw 2D8A Zeichen einfügen

2C75 db FC
2C76 dw 2CD0 ESC

2C78 db EF
2C79 dw 2CCE kein Effekt

2C7B db 0D
2C7C dw 2CF2 ENTER

2C7E db F0
2C7F dw 2D3C CRSR UP (Puffer)

2C81 db F1
2C82 dw 2D0A CRSR DWN (Puffer)

2C84 db F2
2C85 dw 2D34 CRSR LEFT (Puffer)

2C87 db F3
2C88 dw 2D02 CRSR RGHT (Puffer)

2C8A db F8
2C8B dw 2D4F CTRL & CRSR UP

2C8D db F9
2C8E dw 2D1D CTRL & CRSR DWN

2C90 db FA
2C91 dw 2D45 CTRL & CRSR LEFT

```
2C93    db FB
2C94    dw 2D14   CTRL & CRSR RGHT

2C96    db F4
2C97    dw 2E21   SHFT & CRSR UP

2C99    db F5
2C9A    dw 2E26   SHFT & CRSR DWN

2C9C    db F6
2C9D    dw 2E1C   SHFT & CRSR LEFT

2C9F    db F7
2CA0    dw 2E17   SHFT & CRSR RGHT

2CA2    db E0
2CA3    dw 2E65   COPY

2CA5    db 7F
2CA6    dw 2DC3   DEL

2CA8    db 10
2CA9    dw 2DCD   CLR

2CAB    db E1
2CAC    dw 2D81   CTRL & TAB (Flip Insert)

2CAE *********************************** EDIT Sprungtabelle 2

2CAE    db 04   Anzahl Einträge

2CAF    dw 2CFE   Kingel

2CB1    db F0
2CB2    dw 2CBD   CRSR UP

2CB4    db F1
2CB5    dw 2CC1   CRSR DWN
```

2CB7 db F2
2CB8 dw 2CC9 CRSR LEFT

2CBA db F3
2CBB dw 2CC5 CRSR RGHT

2CBD *** CRSR UP

2CC1 *** CRSR DWN

2CC5 *** CRSR RGHT

2CC9 *** CRSR LEFT

2CCB TXT OUTPUT

2CD0 *** ESC

2CD0 ENTER
2CD4 *BREAK*-Meldung
2CD7 ENTER
2CDA TXT GET CURSOR
2CE0 CR
2CE2 TXT OUTPUT
2CE5 CRSR DWN

2CEA ************************************ *BREAK*-Meldung

2CEA 2A 42 72 65 61 6B 2A 00 *BREAK*

2CF1 ** ENTER

2CFC Setze das Übertragsflag

2CFE *** KLINGEL

2CFE BEL

2D02 *********************************** CRSR RGHT (Puffer)

2D07 KLINGEL

2D0A *********************************** CRSR DWN (Puffer)

2D10 KLINGEL

2D14 ********************************** CTRL & CRSR RGHT

2D1D ********************************** CTRL & CRSR DWN

2D34 *********************************** CRSR LEFT (Puffer)

2D39 KLINGEL

2D3C ************************************* CRSR UP (Puffer)

2D41 KLINGEL

2D45 ********************************** CTRL & CRSR LEFT

2D4F ************************************* CTRL & CRSR UP

2D74 TXT GET WINDOW
2D7B TXT GET CURSOR

2D81 ****************************** CTRL & TAB (Filp Insert)

2D81 (Insert Flag)
2D85 (Insert Flag)

2D8A ** Zeichen einfügen

2D8D (Insert Flag)
2DA1 KLINGEL

2DC3 *** DEL

2DC8 KLINGEL

2DCD *** CLR

2DCF KLINGEL
2E0E TXT VALIDATE

2E17 *********************************** SHFT & CRSR RGHT

2E1C *********************************** SHFT & CRSR LEFT

2E21 ************************************** SHFT & CRSR UP

2E26 ************************************* SHFT & CRSR DWN

2E2E TXT GET CURSOR
2E37 TXT VALIDATE
2E4A TXT PLACE/REMOVE CURSOR
2E4F TXT PLACE/REMOVE CURSOR
2E57 TXT GET CURSOR
2E5B TXT SET CURSOR
2E62 TXT SET CURSOR

2E65 *** COPY

2E67 TXT GET CURSOR
2E74 TXT GET CURSOR
2E7C TXT SET CURSOR
2E7F TXT PLACE/REMOVE CURSOR

2E82	TXT RD CHAR
2E87	TXT SET CURSOR
2E8E	TXT VALIDATE
2E9C	Zeichen einfügen
2E9F	KINGEL
2ED3	TXT GET CURSOR
2ED9	TXT VALIDATE
2EDD	TXT OUTPUT
2EE7	TXT GET CURSOR
2EF4	TXT GET CURSOR
2EFB	TXT SET CURSOR
2F07	TXT GET CURSOR
2F0E	TXT VALIDATE
2F19	TXT VALIDATE
2F2A	TXT GET CURSOR
2F2F	TXT VALIDATE
2F3C	TXT WR CHAR
2F40	TXT GET CURSOR

2F56 ********************************** Zeichen von Keyboard

2F56	TXT GET CURSOR
2F5A	TXT VALIDATE
2F60	KM WAIT CHAR
2F63	TXT CUR ON
2F66	TXT GET CURSOR
2F6D	KM WAIT CHAR
2F70	TXT CUR OFF

2.6 Der Character-Generator

Nicht, daß wir der Meinung sind, das Buch sei einfach noch nicht umfangreich genug oder daß wir Sie mit den folgenden Seiten langweilen wollen - was wir glauben, ist einfach, daß der Zeichensatz ein wichtiges Betriebsmittel ist, welchem sogar im BASIC-Befehlsvorrat eigene Kommandos zugestanden werden.

Damit Sie bei deren Anwendungen nicht jedesmal das Rad neu erfinden müssen, z.B. bei der Erzeugung von Umlauten, brauchen Sie sich nur das 'a' herauszusuchen und die beiden Pünktchen zu ergänzen. Die so gefundenen Werte setzen Sie dann in Ihren Befehl ein.

Warum es überhaupt relevant ist, daß Sie sich möglichst an den bereits vorhandenen Zeichen orientieren, ist schnell erklärt:

Es wird Ihnen sicher ins Auge fallen, daß alle vertikalen Linienelemente aus wenigstens zwei benachbarten Punkten zusammengesetzt sind. Der Grund dafür ist die Tatsache, daß sich ein einzelnes Pixel allein auf dem Bildschirm schwerlich wiederfinden ließe. Dieses Phänomen tritt bei Farbmonitoren noch stärker zu Tage als bei Grünmonitoren, weil dort die Schlitzmaske im Wege ist, auf deren Stege dieser Punkt zufällig treffen könnte.

Ziehen Sie also aus dem vorangegangenen Geplänkel die Lehre für Ihre selbstdefinierten Zeichen, bei senkrechten Linien Pixels immer paarweise zu verwenden. Doch sehen Sie erst einmal nach, ob Sie kein brauchbares Zeichen unter den 256 im **CHARACTER GENERATOR** des CPC 664/6128 verfügbaren finden.

CHARACTERS

3800	FF
3801	C3
3802	C3
3803	C3
3804	C3
3805	C3
3806	C3
3807	FF
3808	FF
3809	C0
380A	C0
380B	C0
380C	C0
380D	C0
380E	C0
380F	C0
3810	18
3811	18
3812	18
3813	18
3814	18
3815	18
3816	18
3817	FF
3818	03
3819	03
381A	03
381B	03
381C	03
381D	03
381E	03
381F	FF
3820	0C
3821	18
3822	30
3823	7E
3824	0C
3825	18
3826	30
3827	00
3828	FF
3829	C3
382A	E7
382B	DB
382C	DB
382D	E7
382E	C3
382F	FF
3830	00
3831	01
3832	03
3833	06
3834	CC
3835	78
3836	30
3837	00
3838	3C
3839	66
383A	C3
383B	C3
383C	FF
383D	24
383E	E7
383F	00
3840	00
3841	00
3842	30
3843	60
3844	FF
3845	60
3846	30
3847	00
3848	00
3849	00
384A	0C
384B	06
384C	FF
384D	06
384E	0C
384F	00
3850	18
3851	18
3852	18
3853	18
3854	DB
3855	7E
3856	3C
3857	18
3858	18
3859	3C
385A	7E
385B	DB
385C	18
385D	18
385E	18
385F	18

CHARACTERS

3860	18	3868	00
3861	5A	3869	03
3862	3C	386A	33
3863	99	386B	63
3864	DB	386C	FE
3865	7E	386D	60
3866	3C	386E	30
3867	18	386F	00
3870	3C	3878	3C
3871	66	3879	66
3872	FF	387A	C3
3873	DB	387B	DB
3874	DB	387C	DB
3875	FF	387D	C3
3876	66	387E	66
3877	3C	387F	3C
3880	FF	3888	3C
3881	C3	3889	7E
3882	C3	388A	DB
3883	FF	388B	DB
3884	C3	388C	DF
3885	C3	388D	C3
3886	C3	388E	66
3887	FF	388F	3C
3890	3C	3898	3C
3891	66	3899	66
3892	C3	389A	C3
3893	DF	389B	FB
3894	DB	389C	DB
3895	DB	389D	DB
3896	7E	389E	7E
3897	3C	389F	3C
38A0	3C	38A8	00
38A1	7E	38A9	01
38A2	DB	38AA	33
38A3	DB	38AB	1E
38A4	FB	38AC	CE
38A5	C3	38AD	7B
38A6	66	38AE	31
38A7	3C	38AF	00
38B0	7E	38B8	03
38B1	66	38B9	03
38B2	66	38BA	03
38B3	66	38BB	FF
38B4	66	38BC	03
38B5	66	38BD	03
38B6	66	38BE	03
38B7	E7	38BF	00

CHARACTERS

38C0	FF
38C1	66
38C2	3C
38C3	18
38C4	18
38C5	3C
38C6	66
38C7	FF
38C8	18
38C9	18
38CA	3C
38CB	3C
38CC	3C
38CD	3C
38CE	18
38CF	18
38D0	3C
38D1	66
38D2	66
38D3	30
38D4	18
38D5	00
38D6	18
38D7	00
38D8	3C
38D9	66
38DA	C3
38DB	FF
38DC	C3
38DD	C3
38DE	66
38DF	3C
38E0	FF
38E1	DB
38E2	DB
38E3	DB
38E4	FB
38E5	C3
38E6	C3
38E7	FF
38E8	FF
38E9	C3
38EA	C3
38EB	FB
38EC	DB
38ED	DB
38EE	DB
38EF	FF
38F0	FF
38F1	C3
38F2	C3
38F3	DF
38F4	DB
38F5	DB
38F6	DB
38F7	FF
38F8	FF
38F9	DB
38FA	DB
38FB	DB
38FC	DF
38FD	C3
38FE	C3
38FF	FF
3900	00
3901	00
3902	00
3903	00
3904	00
3905	00
3906	00
3907	00
3908	18
3909	18
390A	18
390B	18
390C	18
390D	00
390E	18
390F	00
3910	6C
3911	6C
3912	6C
3913	00
3914	00
3915	00
3916	00
3917	00
3918	6C
3919	6C
391A	FE
391B	6C
391C	FE
391D	6C
391E	6C
391F	00

CHARACTERS

3920	18	3928	00
3921	3E	3929	C6
3922	58	392A	CC
3923	3C	392B	18
3924	1A	392C	30
3925	7C	392D	66
3926	18	392E	C6
3927	00	392F	00
3930	38	3938	18
3931	6C	3939	18
3932	38	393A	30
3933	76	393B	00
3934	DC	393C	00
3935	CC	393D	00
3936	76	393E	00
3937	00	393F	00
3940	0C	3948	30
3941	18	3949	18
3942	30	394A	0C
3943	30	394B	0C
3944	30	394C	0C
3945	18	394D	18
3946	0C	394E	30
3947	00	394F	00
3950	00	3958	00
3951	66	3959	18
3952	3C	395A	18
3953	FF	395B	7E
3954	3C	395C	18
3955	66	395D	18
3956	00	395E	00
3957	00	395F	00
3960	00	3968	00
3961	00	3969	00
3962	00	396A	00
3963	00	396B	7E
3964	00	396C	00
3965	18	396D	00
3966	18	396E	00
3967	30	396F	00
3970	00	3978	06
3971	00	3979	0C
3972	00	397A	18
3973	00	397B	30
3974	00	397C	60
3975	18	397D	C0
3976	18	397E	80
3977	00	397F	00

CHARACTERS

3980	7C
3981	C6
3982	CE
3983	D6
3984	E6
3985	C6
3986	7C
3987	00

3988	18
3989	38
398A	18
398B	18
398C	18
398D	18
398E	7E
398F	00

3990	3C
3991	66
3992	06
3993	3C
3994	60
3995	66
3996	7E
3997	00

3998	3C
3999	66
399A	06
399B	1C
399C	06
399D	66
399E	3C
399F	00

39A0	1C
39A1	3C
39A2	6C
39A3	CC
39A4	FE
39A5	0C
39A6	1E
39A7	00

39A8	7E
39A9	62
39AA	60
39AB	7C
39AC	06
39AD	66
39AE	3C
39AF	00

39B0	3C
39B1	66
39B2	60
39B3	7C
39B4	66
39B5	66
39B6	3C
39B7	00

39B8	7E
39B9	66
39BA	06
39BB	0C
39BC	18
39BD	18
39BE	18
39BF	00

39C0	3C
39C1	66
39C2	66
39C3	3C
39C4	66
39C5	66
39C6	3C
39C7	00

39C8	3C
39C9	66
39CA	66
39CB	3E
39CC	06
39CD	66
39CE	3C
39CF	00

39D0	00
39D1	00
39D2	18
39D3	18
39D4	00
39D5	18
39D6	18
39D7	00

39D8	00
39D9	00
39DA	18
39DB	18
39DC	00
39DD	18
39DE	18
39DF	30

CHARACTERS

39E0	0C	39E8	00
39E1	18	39E9	00
39E2	30	39EA	7E
39E3	60	39EB	00
39E4	30	39EC	00
39E5	18	39ED	7E
39E6	0C	39EE	00
39E7	00	39EF	00
39F0	60	39F8	3C
39F1	30	39F9	66
39F2	18	39FA	66
39F3	0C	39FB	0C
39F4	18	39FC	18
39F5	30	39FD	00
39F6	60	39FE	18
39F7	00	39FF	00
3A00	7C	3A08	18
3A01	C6	3A09	3C
3A02	DE	3A0A	66
3A03	DE	3A0B	66
3A04	DE	3A0C	7E
3A05	C0	3A0D	66
3A06	7C	3A0E	66
3A07	00	3A0F	00
3A10	FC	3A18	3C
3A11	66	3A19	66
3A12	66	3A1A	C0
3A13	7C	3A1B	C0
3A14	66	3A1C	C0
3A15	66	3A1D	66
3A16	FC	3A1E	3C
3A17	00	3A1F	00
3A20	F8	3A28	FE
3A21	6C	3A29	62
3A22	66	3A2A	68
3A23	66	3A2B	78
3A24	66	3A2C	68
3A25	6C	3A2D	62
3A26	F8	3A2E	FE
3A27	00	3A2F	00
3A30	FE	3A38	3C
3A31	62	3A39	66
3A32	68	3A3A	C0
3A33	78	3A3B	C0
3A34	68	3A3C	CE
3A35	60	3A3D	66
3A36	F0	3A3E	3E
3A37	00	3A3F	00

CHARACTERS

3A40	66
3A41	66
3A42	66
3A43	7E
3A44	66
3A45	66
3A46	66
3A47	00
3A48	7E
3A49	18
3A4A	18
3A4B	18
3A4C	18
3A4D	18
3A4E	7E
3A4F	00
3A50	1E
3A51	0C
3A52	0C
3A53	0C
3A54	CC
3A55	CC
3A56	78
3A57	00
3A58	E6
3A59	66
3A5A	6C
3A5B	78
3A5C	6C
3A5D	66
3A5E	E6
3A5F	00
3A60	F0
3A61	60
3A62	60
3A63	60
3A64	62
3A65	66
3A66	FE
3A67	00
3A68	C6
3A69	EE
3A6A	FE
3A6B	FE
3A6C	D6
3A6D	C6
3A6E	C6
3A6F	00
3A70	C6
3A71	E6
3A72	F6
3A73	DE
3A74	CE
3A75	C6
3A76	C6
3A77	00
3A78	38
3A79	6C
3A7A	C6
3A7B	C6
3A7C	C6
3A7D	6C
3A7E	38
3A7F	00
3A80	FC
3A81	66
3A82	66
3A83	7C
3A84	60
3A85	60
3A86	F0
3A87	00
3A88	38
3A89	6C
3A8A	C6
3A8B	C6
3A8C	DA
3A8D	CC
3A8E	76
3A8F	00
3A90	FC
3A91	66
3A92	66
3A93	7C
3A94	6C
3A95	66
3A96	E6
3A97	00
3A98	3C
3A99	66
3A9A	60
3A9B	3C
3A9C	06
3A9D	66
3A9E	3C
3A9F	00

CHARACTERS

3AA0	7E	3AA8	66
3AA1	5A	3AA9	66
3AA2	18	3AAA	66
3AA3	18	3AAB	66
3AA4	18	3AAC	66
3AA5	18	3AAD	66
3AA6	3C	3AAE	3C
3AA7	00	3AAF	00
3AB0	66	3AB8	C6
3AB1	66	3AB9	C6
3AB2	66	3ABA	C6
3AB3	66	3ABB	D6
3AB4	66	3ABC	FE
3AB5	3C	3ABD	EE
3AB6	18	3ABE	C6
3AB7	00	3ABF	00
3AC0	C6	3AC8	66
3AC1	6C	3AC9	66
3AC2	38	3ACA	66
3AC3	38	3ACB	3C
3AC4	6C	3ACC	18
3AC5	C6	3ACD	18
3AC6	C6	3ACE	3C
3AC7	00	3ACF	00
3AD0	FE	3AD8	3C
3AD1	C6	3AD9	30
3AD2	8C	3ADA	30
3AD3	18	3ADB	30
3AD4	32	3ADC	30
3AD5	66	3ADD	30
3AD6	FE	3ADE	3C
3AD7	00	3ADF	00
3AE0	C0	3AE8	3C
3AE1	60	3AE9	0C
3AE2	30	3AEA	0C
3AE3	18	3AEB	0C
3AE4	0C	3AEC	0C
3AE5	06	3AED	0C
3AE6	02	3AEE	3C
3AE7	00	3AEF	00
3AF0	18	3AF8	00
3AF1	3C	3AF9	00
3AF2	7E	3AFA	00
3AF3	18	3AFB	00
3AF4	18	3AFC	00
3AF5	18	3AFD	00
3AF6	18	3AFE	00
3AF7	00	3AFF	FF

CHARACTERS

3B00	30
3B01	18
3B02	0C
3B03	00
3B04	00
3B05	00
3B06	00
3B07	00
3B08	00
3B09	00
3B0A	78
3B0B	0C
3B0C	7C
3B0D	CC
3B0E	76
3B0F	00
3B10	E0
3B11	60
3B12	7C
3B13	66
3B14	66
3B15	66
3B16	DC
3B17	00
3B18	00
3B19	00
3B1A	3C
3B1B	66
3B1C	60
3B1D	66
3B1E	3C
3B1F	00
3B20	1C
3B21	0C
3B22	7C
3B23	CC
3B24	CC
3B25	CC
3B26	76
3B27	00
3B28	00
3B29	00
3B2A	3C
3B2B	66
3B2C	7E
3B2D	60
3B2E	3C
3B2F	00
3B30	1C
3B31	36
3B32	30
3B33	78
3B34	30
3B35	30
3B36	78
3B37	00
3B38	00
3B39	00
3B3A	3E
3B3B	66
3B3C	66
3B3D	3E
3B3E	06
3B3F	7C
3B40	E0
3B41	60
3B42	6C
3B43	76
3B44	66
3B45	66
3B46	E6
3B47	00
3B48	18
3B49	00
3B4A	38
3B4B	18
3B4C	18
3B4D	18
3B4E	3C
3B4F	00
3B50	06
3B51	00
3B52	0E
3B53	06
3B54	06
3B55	66
3B56	66
3B57	3C
3B58	E0
3B59	60
3B5A	66
3B5B	6C
3B5C	78
3B5D	6C
3B5E	E6
3B5F	00

CHARACTERS

3B60	38
3B61	18
3B62	18
3B63	18
3B64	18
3B65	18
3B66	3C
3B67	00
3B68	00
3B69	00
3B6A	6C
3B6B	FE
3B6C	D6
3B6D	D6
3B6E	C6
3B6F	00
3B70	00
3B71	00
3B72	DC
3B73	66
3B74	66
3B75	66
3B76	66
3B77	00
3B78	00
3B79	00
3B7A	3C
3B7B	66
3B7C	66
3B7D	66
3B7E	3C
3B7F	00
3B80	00
3B81	00
3B82	DC
3B83	66
3B84	66
3B85	7C
3B86	60
3B87	F0
3B88	00
3B89	00
3B8A	76
3B8B	CC
3B8C	CC
3B8D	7C
3B8E	0C
3B8F	1E
3B90	00
3B91	00
3B92	DC
3B93	76
3B94	60
3B95	60
3B96	F0
3B97	00
3B98	00
3B99	00
3B9A	3C
3B9B	60
3B9C	3C
3B9D	06
3B9E	7C
3B9F	00
3BA0	30
3BA1	30
3BA2	7C
3BA3	30
3BA4	30
3BA5	36
3BA6	1C
3BA7	00
3BA8	00
3BA9	00
3BAA	66
3BAB	66
3BAC	66
3BAD	66
3BAE	3E
3BAF	00
3BB0	00
3BB1	00
3BB2	66
3BB3	66
3BB4	66
3BB5	3C
3BB6	18
3BB7	00
3BB8	00
3BB9	00
3BBA	C6
3BBB	D6
3BBC	D6
3BBD	FE
3BBE	6C
3BBF	00

CHARACTERS

3BC0	00
3BC1	00
3BC2	C6
3BC3	6C
3BC4	38
3BC5	6C
3BC6	C6
3BC7	00
3BD0	00
3BD1	00
3BD2	7E
3BD3	4C
3BD4	18
3BD5	32
3BD6	7E
3BD7	00
3BE0	18
3BE1	18
3BE2	18
3BE3	18
3BE4	18
3BE5	18
3BE6	18
3BE7	00
3BF0	76
3BF1	DC
3BF2	00
3BF3	00
3BF4	00
3BF5	00
3BF6	00
3BF7	00
3C00	00
3C01	00
3C02	00
3C03	00
3C04	00
3C05	00
3C06	00
3C07	00
3C10	0F
3C11	0F
3C12	0F
3C13	0F
3C14	00
3C15	00
3C16	00
3C17	00

3BC8	00
3BC9	00
3BCA	66
3BCB	66
3BCC	66
3BCD	3E
3BCE	06
3BCF	7C
3BD8	0E
3BD9	18
3BDA	18
3BDB	70
3BDC	18
3BDD	18
3BDE	0E
3BDF	00
3BE8	70
3BE9	18
3BEA	18
3BEB	0E
3BEC	18
3BED	18
3BEE	70
3BEF	00
3BF8	CC
3BF9	33
3BFA	CC
3BFB	33
3BFC	CC
3BFD	33
3BFE	CC
3BFF	33
3C08	F0
3C09	F0
3C0A	F0
3C0B	F0
3C0C	00
3C0D	00
3C0E	00
3C0F	00
3C18	FF
3C19	FF
3C1A	FF
3C1B	FF
3C1C	00
3C1D	00
3C1E	00
3C1F	00

CHARACTERS

3C20	00	3C28	F0
3C21	00	3C29	F0
3C22	00	3C2A	F0
3C23	00	3C2B	F0
3C24	F0	3C2C	F0
3C25	F0	3C2D	F0
3C26	F0	3C2E	F0
3C27	F0	3C2F	F0
3C30	0F	3C38	FF
3C31	0F	3C39	FF
3C32	0F	3C3A	FF
3C33	0F	3C3B	FF
3C34	F0	3C3C	F0
3C35	F0	3C3D	F0
3C36	F0	3C3E	F0
3C37	F0	3C3F	F0
3C40	00	3C48	F0
3C41	00	3C49	F0
3C42	00	3C4A	F0
3C43	00	3C4B	F0
3C44	0F	3C4C	0F
3C45	0F	3C4D	0F
3C46	0F	3C4E	0F
3C47	0F	3C4F	0F
3C50	0F	3C58	FF
3C51	0F	3C59	FF
3C52	0F	3C5A	FF
3C53	0F	3C5B	FF
3C54	0F	3C5C	0F
3C55	0F	3C5D	0F
3C56	0F	3C5E	0F
3C57	0F	3C5F	0F
3C60	00	3C68	F0
3C61	00	3C69	F0
3C62	00	3C6A	F0
3C63	00	3C6B	F0
3C64	FF	3C6C	FF
3C65	FF	3C6D	FF
3C66	FF	3C6E	FF
3C67	FF	3C6F	FF
3C70	0F	3C78	FF
3C71	0F	3C79	FF
3C72	0F	3C7A	FF
3C73	0F	3C7B	FF
3C74	FF	3C7C	FF
3C75	FF	3C7D	FF
3C76	FF	3C7E	FF
3C77	FF	3C7F	FF

CHARACTERS

3C80	00	3C88	18
3C81	00	3C89	18
3C82	00	3C8A	18
3C83	18	3C8B	18
3C84	18	3C8C	18
3C85	00	3C8D	00
3C86	00	3C8E	00
3C87	00	3C8F	00
3C90	00	3C98	18
3C91	00	3C99	18
3C92	00	3C9A	18
3C93	1F	3C9B	1F
3C94	1F	3C9C	0F
3C95	00	3C9D	00
3C96	00	3C9E	00
3C97	00	3C9F	00
3CA0	00	3CA8	18
3CA1	00	3CA9	18
3CA2	00	3CAA	18
3CA3	18	3CAB	18
3CA4	18	3CAC	18
3CA5	18	3CAD	18
3CA6	18	3CAE	18
3CA7	18	3CAF	18
3CB0	00	3CB8	18
3CB1	00	3CB9	18
3CB2	00	3CBA	18
3CB3	0F	3CBB	1F
3CB4	1F	3CBC	1F
3CB5	18	3CBD	18
3CB6	18	3CBE	18
3CB7	18	3CBF	18
3CC0	00	3CC8	18
3CC1	00	3CC9	18
3CC2	00	3CCA	18
3CC3	F8	3CCB	F8
3CC4	F8	3CCC	F0
3CC5	00	3CCD	00
3CC6	00	3CCE	00
3CC7	00	3CCF	00
3CD0	00	3CD8	18
3CD1	00	3CD9	18
3CD2	00	3CDA	18
3CD3	FF	3CDB	FF
3CD4	FF	3CDC	FF
3CD5	00	3CDD	00
3CD6	00	3CDE	00
3CD7	00	3CDF	00

CHARACTERS

3CE0	00	3CE8	18
3CE1	00	3CE9	18
3CE2	00	3CEA	18
3CE3	F0	3CEB	F8
3CE4	F8	3CEC	F8
3CE5	18	3CED	18
3CE6	18	3CEE	18
3CE7	18	3CEF	18

3CF0	00	3CF8	18
3CF1	00	3CF9	18
3CF2	00	3CFA	18
3CF3	FF	3CFB	FF
3CF4	FF	3CFC	FF
3CF5	18	3CFD	18
3CF6	18	3CFE	18
3CF7	18	3CFF	18

3D00	10	3D08	0C
3D01	38	3D09	18
3D02	6C	3D0A	30
3D03	C6	3D0B	00
3D04	00	3D0C	00
3D05	00	3D0D	00
3D06	00	3D0E	00
3D07	00	3D0F	00

3D10	66	3D18	3C
3D11	66	3D19	66
3D12	00	3D1A	60
3D13	00	3D1B	F8
3D14	00	3D1C	60
3D15	00	3D1D	66
3D16	00	3D1E	FE
3D17	00	3D1F	00

3D20	38	3D28	7E
3D21	44	3D29	F4
3D22	BA	3D2A	F4
3D23	A2	3D2B	74
3D24	BA	3D2C	34
3D25	44	3D2D	34
3D26	38	3D2E	34
3D27	00	3D2F	00

3D30	1E	3D38	18
3D31	30	3D39	18
3D32	38	3D3A	0C
3D33	6C	3D3B	00
3D34	38	3D3C	00
3D35	18	3D3D	00
3D36	F0	3D3E	00
3D37	00	3D3F	00

CHARACTERS

3D40	40
3D41	C0
3D42	44
3D43	4C
3D44	54
3D45	1E
3D46	04
3D47	00
3D48	40
3D49	C0
3D4A	4C
3D4B	52
3D4C	44
3D4D	08
3D4E	1E
3D4F	00
3D50	E0
3D51	10
3D52	62
3D53	16
3D54	EA
3D55	0F
3D56	02
3D57	00
3D58	00
3D59	18
3D5A	18
3D5B	7E
3D5C	18
3D5D	18
3D5E	7E
3D5F	00
3D60	18
3D61	18
3D62	00
3D63	7E
3D64	00
3D65	18
3D66	18
3D67	00
3D68	00
3D69	00
3D6A	00
3D6B	7E
3D6C	06
3D6D	06
3D6E	00
3D6F	00
3D70	18
3D71	00
3D72	18
3D73	30
3D74	66
3D75	66
3D76	3C
3D77	00
3D78	18
3D79	00
3D7A	18
3D7B	18
3D7C	18
3D7D	18
3D7E	18
3D7F	00
3D80	00
3D81	00
3D82	73
3D83	DE
3D84	CC
3D85	DE
3D86	73
3D87	00
3D88	7C
3D89	C6
3D8A	C6
3D8B	FC
3D8C	C6
3D8D	C6
3D8E	F8
3D8F	C0
3D90	00
3D91	66
3D92	66
3D93	3C
3D94	66
3D95	66
3D96	3C
3D97	00
3D98	3C
3D99	60
3D9A	60
3D9B	3C
3D9C	66
3D9D	66
3D9E	3C
3D9F	00

CHARACTERS

3DA0	00	3DA8	38
3DA1	00	3DA9	6C
3DA2	1E	3DAA	C6
3DA3	30	3DAB	FE
3DA4	7C	3DAC	C6
3DA5	30	3DAD	6C
3DA6	1E	3DAE	38
3DA7	00	3DAF	00
3DB0	00	3DB8	00
3DB1	C0	3DB9	00
3DB2	60	3DBA	66
3DB3	30	3DBB	66
3DB4	38	3DBC	66
3DB5	6C	3DBD	7C
3DB6	C6	3DBE	60
3DB7	00	3DBF	60
3DC0	00	3DC8	00
3DC1	00	3DC9	00
3DC2	00	3DCA	00
3DC3	FE	3DCB	7E
3DC4	6C	3DCC	D8
3DC5	6C	3DCD	D8
3DC6	6C	3DCE	70
3DC7	00	3DCF	00
3DD0	03	3DD8	03
3DD1	06	3DD9	06
3DD2	0C	3DDA	0C
3DD3	3C	3DDB	66
3DD4	66	3DDC	66
3DD5	3C	3DDD	3C
3DD6	60	3DDE	60
3DD7	C0	3DDF	C0
3DE0	00	3DE8	00
3DE1	E6	3DE9	00
3DE2	3C	3DEA	66
3DE3	18	3DEB	C3
3DE4	38	3DEC	DB
3DE5	6C	3DED	DB
3DE6	C7	3DEE	7E
3DE7	00	3DEF	00
3DF0	FE	3DF8	00
3DF1	C6	3DF9	7C
3DF2	60	3DFA	C6
3DF3	30	3DFB	C6
3DF4	60	3DFC	C6
3DF5	C6	3DFD	6C
3DF6	FE	3DFE	EE
3DF7	00	3DFF	00

CHARACTERS

3E00	18
3E01	30
3E02	60
3E03	C0
3E04	80
3E05	00
3E06	00
3E07	00

3E08	18
3E09	0C
3E0A	06
3E0B	03
3E0C	01
3E0D	00
3E0E	00
3E0F	00

3E10	00
3E11	00
3E12	00
3E13	01
3E14	03
3E15	06
3E16	0C
3E17	18

3E18	00
3E19	00
3E1A	00
3E1B	80
3E1C	C0
3E1D	60
3E1E	30
3E1F	18

3E20	18
3E21	3C
3E22	66
3E23	C3
3E24	81
3E25	00
3E26	00
3E27	00

3E28	18
3E29	0C
3E2A	06
3E2B	03
3E2C	03
3E2D	06
3E2E	0C
3E2F	18

3E30	00
3E31	00
3E32	00
3E33	81
3E34	C3
3E35	66
3E36	3C
3E37	18

3E38	18
3E39	30
3E3A	60
3E3B	C0
3E3C	C0
3E3D	60
3E3E	30
3E3F	18

3E40	18
3E41	30
3E42	60
3E43	C1
3E44	83
3E45	06
3E46	0C
3E47	18

3E48	18
3E49	0C
3E4A	06
3E4B	83
3E4C	C1
3E4D	60
3E4E	30
3E4F	18

3E50	18
3E51	3C
3E52	66
3E53	C3
3E54	C3
3E55	66
3E56	3C
3E57	18

3E58	C3
3E59	E7
3E5A	7E
3E5B	3C
3E5C	3C
3E5D	7E
3E5E	E7
3E5F	C3

CHARACTERS

3E60	03
3E61	07
3E62	0E
3E63	1C
3E64	38
3E65	70
3E66	E0
3E67	C0
3E68	C0
3E69	E0
3E6A	70
3E6B	38
3E6C	1C
3E6D	0E
3E6E	07
3E6F	03
3E70	CC
3E71	CC
3E72	33
3E73	33
3E74	CC
3E75	CC
3E76	33
3E77	33
3E78	AA
3E79	55
3E7A	AA
3E7B	55
3E7C	AA
3E7D	55
3E7E	AA
3E7F	55
3E80	FF
3E81	FF
3E82	00
3E83	00
3E84	00
3E85	00
3E86	00
3E87	00
3E88	03
3E89	03
3E8A	03
3E8B	03
3E8C	03
3E8D	03
3E8E	03
3E8F	03
3E90	00
3E91	00
3E92	00
3E93	00
3E94	00
3E95	00
3E96	FF
3E97	FF
3E98	C0
3E99	C0
3E9A	C0
3E9B	C0
3E9C	C0
3E9D	C0
3E9E	C0
3E9F	C0
3EA0	FF
3EA1	FE
3EA2	FC
3EA3	F8
3EA4	F0
3EA5	E0
3EA6	C0
3EA7	80
3EA8	FF
3EA9	7F
3EAA	3F
3EAB	1F
3EAC	0F
3EAD	07
3EAE	03
3EAF	01
3EB0	01
3EB1	03
3EB2	07
3EB3	0F
3EB4	1F
3EB5	3F
3EB6	7F
3EB7	FF
3EB8	80
3EB9	C0
3EBA	E0
3EBB	F0
3EBC	F8
3EBD	FC
3EBE	FE
3EBF	FF

CHARACTERS

3EC0	AA	3EC8	0A
3EC1	55	3EC9	05
3EC2	AA	3ECA	0A
3EC3	55	3ECB	05
3EC4	00	3ECC	0A
3EC5	00	3ECD	05
3EC6	00	3ECE	0A
3EC7	00	3ECF	05
3ED0	00	3ED8	A0
3ED1	00	3ED9	50
3ED2	00	3EDA	A0
3ED3	00	3EDB	50
3ED4	AA	3EDC	A0
3ED5	55	3EDD	50
3ED6	AA	3EDE	A0
3ED7	55	3EDF	50
3EE0	AA	3EE8	AA
3EE1	54	3EE9	55
3EE2	A8	3EEA	2A
3EE3	50	3EEB	15
3EE4	A0	3EEC	0A
3EE5	40	3EED	05
3EE6	80	3EEE	02
3EE7	00	3EEF	01
3EF0	01	3EF8	00
3EF1	02	3EF9	80
3EF2	05	3EFA	40
3EF3	0A	3EFB	A0
3EF4	15	3EFC	50
3EF5	2A	3EFD	A8
3EF6	55	3EFE	54
3EF7	AA	3EFF	AA
3F00	7E	3F08	7E
3F01	FF	3F09	FF
3F02	99	3F0A	99
3F03	FF	3F0B	FF
3F04	BD	3F0C	C3
3F05	C3	3F0D	BD
3F06	FF	3F0E	FF
3F07	7E	3F0F	7E
3F10	38	3F18	10
3F11	38	3F19	38
3F12	FE	3F1A	7C
3F13	FE	3F1B	FE
3F14	FE	3F1C	7C
3F15	10	3F1D	38
3F16	38	3F1E	10
3F17	00	3F1F	00

CHARACTERS

3F20	6C	3F28	10
3F21	FE	3F29	38
3F22	FE	3F2A	7C
3F23	FE	3F2B	FE
3F24	7C	3F2C	FE
3F25	38	3F2D	10
3F26	10	3F2E	38
3F27	00	3F2F	00
3F30	00	3F38	00
3F31	3C	3F39	3C
3F32	66	3F3A	7E
3F33	C3	3F3B	FF
3F34	C3	3F3C	FF
3F35	66	3F3D	7E
3F36	3C	3F3E	3C
3F37	00	3F3F	00
3F40	00	3F48	00
3F41	7E	3F49	7E
3F42	66	3F4A	7E
3F43	66	3F4B	7E
3F44	66	3F4C	7E
3F45	66	3F4D	7E
3F46	7E	3F4E	7E
3F47	00	3F4F	00
3F50	0F	3F58	3C
3F51	07	3F59	66
3F52	0D	3F5A	66
3F53	78	3F5B	66
3F54	CC	3F5C	3C
3F55	CC	3F5D	18
3F56	CC	3F5E	7E
3F57	78	3F5F	18
3F60	0C	3F68	18
3F61	0C	3F69	1C
3F62	0C	3F6A	1E
3F63	0C	3F6B	1B
3F64	0C	3F6C	18
3F65	3C	3F6D	78
3F66	7C	3F6E	F8
3F67	38	3F6F	70
3F70	99	3F78	10
3F71	5A	3F79	38
3F72	24	3F7A	38
3F73	C3	3F7B	38
3F74	C3	3F7C	38
3F75	24	3F7D	38
3F76	5A	3F7E	7C
3F77	99	3F7F	D6

CHARACTERS

3F80	18
3F81	3C
3F82	7E
3F83	FF
3F84	18
3F85	18
3F86	18
3F87	18
3F88	18
3F89	18
3F8A	18
3F8B	18
3F8C	FF
3F8D	7E
3F8E	3C
3F8F	18
3F90	10
3F91	30
3F92	70
3F93	FF
3F94	FF
3F95	70
3F96	30
3F97	10
3F98	08
3F99	0C
3F9A	0E
3F9B	FF
3F9C	FF
3F9D	0E
3F9E	0C
3F9F	08
3FA0	00
3FA1	00
3FA2	18
3FA3	3C
3FA4	7E
3FA5	FF
3FA6	FF
3FA7	00
3FA8	00
3FA9	00
3FAA	FF
3FAB	FF
3FAC	7E
3FAD	3C
3FAE	18
3FAF	00
3FB0	80
3FB1	E0
3FB2	F8
3FB3	FE
3FB4	F8
3FB5	E0
3FB6	80
3FB7	00
3FB8	02
3FB9	0E
3FBA	3E
3FBB	FE
3FBC	3E
3FBD	0E
3FBE	02
3FBF	00
3FC0	38
3FC1	38
3FC2	92
3FC3	7C
3FC4	10
3FC5	28
3FC6	28
3FC7	28
3FC8	38
3FC9	38
3FCA	10
3FCB	FE
3FCC	10
3FCD	28
3FCE	44
3FCF	82
3FD0	38
3FD1	38
3FD2	12
3FD3	7C
3FD4	90
3FD5	28
3FD6	24
3FD7	22
3FD8	38
3FD9	38
3FDA	90
3FDB	7C
3FDC	12
3FDD	28
3FDE	48
3FDF	88

CHARACTERS

3FE0	00	3FE8	3C
3FE1	3C	3FE9	FF
3FE2	18	3FEA	FF
3FE3	3C	3FEB	18
3FE4	3C	3FEC	0C
3FE5	3C	3FED	18
3FE6	18	3FEE	30
3FE7	00	3FEF	18

3FF0	18	3FF8	00
3FF1	3C	3FF9	24
3FF2	7E	3FFA	66
3FF3	18	3FFB	FF
3FF4	18	3FFC	66
3FF5	7E	3FFD	24
3FF6	3C	3FFE	00
3FF7	18	3FFF	00

3 BASIC

3.1 Der BASIC-Interpreter des CPC 664 & CPC 6128

Der CPC hat einen schnellen und komfortablen BASIC-Interpreter, der in 16 KByte ROM untergebracht ist und für beide Rechner gleich ist. Er belegt den Adreßbereich von &C000 bis &FFFF parallel zum Bildschirm-RAM. Für BASIC-Programm und -Variablen steht der Bereich von &0170 bis &A67B zur Verfügung, das sind 42249 Byte.

Der Interpreter unterstützt fast alle Möglichkeiten, die von der Hardware und dem Betriebssystem her gegeben sind. Dazu gehören vor allem die Bildschirmausgabe mit bis zu 8 Fenstern, die hochauflösende Grafik, der Sound sowie die Eventverarbeitung. Damit ist es erstmals von BASIC aus möglich, mehrere 'Jobs' nebeneinander laufen zu lassen. Desweiteren bietet der BASIC-Interpreter eine Integerarithmetik mit 16-Bit-Zahlen (Wertebereich -32768 bis 32767) und eine Fließkommaarithmetik mit 8-Bit-Zweierexponent und 32-Bit-Mantisse, die eine Genauigkeit von 9 Dezimalstellen bei einem Wertebereich von +/-1E-39 bis +/-1E+38 garantiert.

Dabei ist die Fließkommaarithmetik nicht Teil des BASIC-Interpreters, sondern im Betriebssystem-ROM enthalten (Adresse &2F73 bis &37FF). Sie wird ebenso wie die übrigen Funktionen des Betriebssystems über die Sprungtabelle im oberen RAM-Bereich (&BB00 bis &BDF4) aufgerufen, die im Bedarfsfall geändert werden kann.

Auch sonst ermöglicht der BASIC-Interpreter ein komfortables Erstellen, Editieren und Ausführen von Programmen. Dem ersteren dient der Befehl AUTO, zum Editieren der EDIT-Befehl, der durch die Fähigkeiten des Betriebssystems kaum dem Arbeiten mit einem Bildschirmeditor nachsteht, sowie die Befehle RENUM, MERGE und DELETE. Auch bei der Ausführung des Programms braucht auf Komfort nicht verzichtet werden. Als Beispiel mögen die Fehlerbehandlung mit ON ERROR GOTO,

die Typdefinition von Variablen mit DEFtyp, das selektive Löschen von Feldern mit ERASE, die Ein- und Ausgabe von Zahlen als Dezimal-, Binär- oder Hexzahl, selbstdefinierte Funktionen mit mehreren Argumenten und allen Datentypen und Programmstrukturen wie IF ... THEN ... ELSE, FOR ... NEXT und WHILE ... WEND dienen. Die Neubelegung von Tasten und Funktionstasten ist ebenso einfach möglich wie die Definition eigener Sonderzeichen auf dem Bildschirm. Ein TRACE-Befehl fehlt ebensowenig wie ein umfassendes PRINT USING.

Nach dieser kurzen Übersicht soll auf die Eingabe und Ablage von BASIC-Zeilen sowie die Ausführung durch den BASIC-Interpreter etwas näher eingegangen werden. Mit diesen Kenntnissen können Sie nicht nur 'das Letzte' aus dem BASIC-Interpreter herausholen, sie sind auch die Grundlage zum Schreiben eigener BASIC-Erweiterungen, von denen später noch einige Beispiele folgen.

Die Eingabe von BASIC-Zeilen

Wenn Sie eine BASIC-Zeile eingeben, so wird sie zuerst in einen 256 Byte großen Puffer übernommen, der von Adresse &ACA8 bis &ADA7 liegt. Dort steht die Eingabe im Klartext. Beginnt die Zeile mit einer Zeilennummer, so wird diese in eine 16-Bit-Binärzahl ungewandelt und in einen zweiten Puffer für die umgewandelte Zeile abgelegt. Dieser Puffer ist 300 Zeichen groß und liegt vor dem BASIC-Programm von Adresse &40 bis &16F. Dann wird die Eingabezeile auf BASIC-Schlüsselworte durchsucht. Diese Schlüsselworte werden durch ein Byte ersetzt, das sogenannte Token. So wird z.B aus 'AFTER' das Token &80. Die Tokens sämtlicher Befehlsworte und der BASIC-Operatoren wie '=' und 'AND' haben Werte größer als 127, das 7. Bit ist also gesetzt. BASIC-Funktionen wie EXP oder ROUND haben Tokens zwischen 0 und &7F. Um sie von normalen ASCII-Zeichen zu unterscheiden, werden sie durch ein vorangestelltes &FF gekennzeichnet. Der Doppelpunkt zum Trennen zweier Statements wird durch den Kode &01 dargestellt, das Zeilenende wird durch &00

abgeschlossen. Konnte eine Buchstabenfolge nicht als Befehl oder Funktion identifiziert werden, so wird sie als Variablenname behandelt. Eine Variablenname kann bis zu 40 Zeichen lang sein, die alle signifikant sind. Zwischen Klein- und Großschreibung wird dabei nicht unterschieden. Nehmen wir an, wir hätten folgende Zeile eingegeben:

```
start=77
```

Nach der Zeilennummer wird jetzt folgendes abgelegt:

```
&0D &00 &00 &73 &74 &61 &72 &F4 &EF &19 &4D &00ü
```

Dabei bedeutet die &0D, daß es sich um eine Variable ohne Typkennzeichen handelt. Dann folgen zwei Nullbytes, auf die wir später noch zurück kommen. Nun folgt der Name der Variablen, die ASCII-Codes für s,t,a und r. Beim letzten Zeichen 't' wird zum ASCII-Code &74 noch &80 dazu addiert (das oberste Bit wird gesetzt) und wir erhalten &F4. Der Code &EF ist das Token für '='. Das nachfolgende &19 kündigt eine Ein-Byte-Konstante an: &4D ist gleich 77. Die abschließende Null bedeutet das Ende der Zeile.

Vor der Zeilennummer folgen noch zwei Bytes, die die Länge der Zeile angeben:

```
&12 &00 &0A &00ü
```

Die Zeile ist also &12 + 256 * &00 gleich 18 Bytes lang und hat die Zeilennummer &0A + 256 * &00 gleich 10.

Sie sehen also, daß im Gegensatz zu anderen BASIC-Interpretern Konstanten im Programmtext nicht als ASCII-Texte abgelegt, sondern bereits in die Binärform überführt worden sind. Dies hat einen ganz entscheidenen Vorteil. Die zeitintensive Umwandlung vom ASCII ins Binärformat braucht nur einmal, nämlich bei der Eingabe der Zeile zu geschehen und nicht bei jedem Ausführen der Zeile noch einmal. Dies bedeutet einen nicht unerheblichen Geschwindigkeitsvorteil beim Ausführen der Programme.

Der CPC kennt noch eine ganze Reihe von numerischen Konstanten, die durch ein entsprechendes Token gekennzeichnet sind. Konstanten, die nur aus einer Ziffer bestehen, also die Zahlen von 0 bis 9, werden durch die Token &0E bis &17 kodiert, belegen also nur ein Byte im Programmtext. Das Token &19 für Ein-Byte-Werte haben wir bereits kenngelernt. Für Zwei-Byte-Integerwerte gibt es drei verschiedene Token, je nachdem, ob die Konstante dezimal, binär oder hex eingegeben wurde. Die Ablage mit Lo- und Hi-Byte ist in allen drei Fällen gleich.

```
&1A Zwei-Byte-Wert, dezimal
&1B Zwei-Byte-Wert, binär
&1C Zwei-Byte-Wert, hexadezimal
```

Handelt es sich um keine ganze Zahl oder ist der Betrag größer als 32767, so wird sie als Fließkommawert abgelegt, der durch das Token &1F gekennzeichnet wird. Dann folgen 5 Bytes, der Fließkommawert. Auf die Fließkommazahlen wird an anderer Stelle noch eingegangen.

Eine Sonderstellung nehmen in diesem Zusammenhang die Zeilennummern ein, wie sie z.B. nach Befehlen wie GOTO, GOSUB oder RUN folgen. Sie werden ebenfalls als 16-Bit-Binärzahl abgelegt, jedoch durch das Token &1E gekennzeichnet.

Wird ein Programm ausgeführt und trifft es z.B. auf einen GOTO-Befehl, so liest es die Zeilennummer und muß das ganze Programm nach dieser Zeile durchsuchen. Besonders bei größeren Programmen kann das relativ lange dauern. Oft werden GOTO- und GOSUB-Befehle in Programmschleifen benutzt und hundert oder tausende Male durchlaufen. Dabei können die Suchzeiten dann einen Großteil der Programmausführungszeit ausmachen. Der BASIC-Interpreter des CPC führt diese Zeilensuche nur einmal durch. Hat er die Zeile gefunden, so ersetzt er die Zeilennummer hinter dem GOTO-Befehl durch die

Adresse der Zeile, die er gefunden hat. Damit er die Adresse von einer Zeilennummer unterscheiden kann, ändert er das Token &1E in &1D, das Token für die Zeilenadresse. Wird der GOTO-Befehl dann nochmal durchgeführt, hat der Interpreter direkt die Adresse, an die die Programmausführung verzweigen kann, was viel Zeit spart.

Dieses Verfahren bringt jedoch bei Befehlen, die die Zeilennummer als solche brauchen, Schwierigkeiten. Wenn der LIST-Befehl die Zeile ausgeben soll, muß er ja die Zeilennummer ausgeben und nicht deren Adresse. Dieses Problem ist jedoch leicht gelöst. Wenn die Zeilenadresse bekannt ist, kann einfach von dort die Zeilennummer geholt werden, da ja, wie wir oben gesehen haben, die Zeilennummer in der Zeile abgelegt ist. Wenn Zeilen gelöscht oder eingefügt werden, muß eine Ersetzung der Zeilennadresse durch die Zeilennumern erfolgen, da die Adressen sich danach ja ändern. Dies betrifft jedoch nur die Ein- und Ausgabe von Programmzeilen und wird durch die bedeutend schnellere Programmausführung bei weitem wieder ausgeglichen.

Die Programmausführung durch den BASIC-Interpreter

Die Ausführung eines Statements durch den BASIC-Interpreter läßt sich vereinfacht folgendermaßen darstellen. Jede Programmzeile beginnt wie beschrieben mit der Programmlänge und der Zeilennummer. Danach kommt der eigentliche BASIC-Befehl. Der Interpreter prüft nun, ob es sich um ein Befehlstoken handelt, das durch einen Wert zwischen &80 und &E1 gekennzeichnet ist. Ist dies der Fall, so benutzt er dieses Token als Zeiger in eine Tabelle, die die Adressen sämtlicher BASIC-Befehle enthält. Der BASIC-Befehl wird als Unterprogramm ausgeführt. Danach wird wieder in die sogenannte Interpreterschleife zurückgekehrt. Begann die Anweisung jedoch nicht mit einem Befehlstoken, so wird zum LET-Befehl verzweigt.

Der wohl wichtigste Teil des BASIC-Interpreters ist die Ausdrucksberechnung. Der CPC unterscheidet dabei zwischen drei Typen von Ausdrücken: Integer, Fließkomma und String. Wenn z.B. eine Wertzuweisung an eine Variable ausgeführt wird oder wenn ein Parameter zu einem Befehl berechnet werden soll, so wird eine Routine aufgerufen, die den Ausdruck berechnet und den Wert sowie den Typ des Ausdrucks bereitstellt. Der Variablentyp kann drei Werte annehmen:

```
2 Integer
3 String
5 Fließkomma
```

Diese Typnummer ist gleichzeitig die Länge der Variablen. Bei einem String ist das der sogenannte Descriptor, der Länge und Adresse enthält (siehe auch Kapitel über den Variablenpointer). Stimmen nun Type eines Ausdrucks und einer Variablen, an die dieser Ausdruck zugewiesen werden soll, nicht überein, so wird versucht, eine Typumwandlung durchzuführen. Das ist jedoch nur bei den numerischen Typen Integer und Fließkomma möglich. Diese Umwandlung kostet natürlich Rechenzeit. Deshalb sollte man immmer dort, wo es möglich ist, Integervariablen einsetzen. Die Praxis hat gezeigt, daß oft in 90% der Fälle Integervariablen eingesetzt werden können. Dadurch entfällt nicht nur eine Typumwandlung, sondern die Integerarithmetik ist auch noch bedeutend schneller als die Fließkommaarithmetik. Ganz besonders sollte dies für die Laufvariablen in FOR-NEXT-Schleifen und sonstige Zähler gelten.

Wird dagegen versucht, einen Stringausdruck einer numerischen Variablen oder umgekehrt zuzuweisen, so wird die Fehlermeldung 'Type mismatch' ausgegeben. Eine Umwandlung von String nach numerisch und umgekehrt ist nur explizit mit den Funktionen VAL und STR$ möglich.

3.2 Der BASIC-Stack

Ein Stack oder Stapelspeicher dient zum Ablegen von Daten nach dem 'Last in - First out' Prinzip. Der Prozessor benutzt dazu den Speicherbereich ab &C000. Vor jedem Eintrag wird der Stackpointer (Stapelzeiger) dekrementiert. Werden Daten wieder vom Stack geholt, so wird anschließend der Stapelzeiger wieder inkrementiert. Der Prozessorstack dient z.B. zur Ablage von Rücksprungadressen bei Unterprogrammaufrufen und erlaubt durch das Zugriffsprinzip eine Schachtelung von Unterprogrammen.

Der BASIC-Interpreter braucht zur Ablage der Parameter von GOSUB-Aufrufen, FOR-NEXT- und WHILE-WEND-Schleifen ebenfalls einen Stack, der erst eine Schachtelung dieser Programmstrukturen möglich macht. Dazu wird nun nicht der Prozessorstack mitbenutzt, sondern es existiert ein eigener BASIC-Stack, der 512 Bytes groß ist und bei Adresse &AE8B beginnt. Im Gegensatz zum Prozessorstack wächst dieser Stack mit zunehmenden Einträgen zu höheren Adressen hin bis maximal &B08A. Als Stackpointer dient die Speicherstelle &B08B/&B08C.

Betrachten wir zuerst, welche Parameter bei einem GOSUB-Befehl auf dem Stack abgelegt werden.

&00/&01	**Kennzeichen der GOSUB-Klasse**
Lo Hi	**Adresse der Anweisung nach dem GOSUB-Befehl**
Lo Hi	**Zeilenadresse der GOSUB-Anweisung**
&06	**Größe des Stackeintrags**

Zuerst wird also ein Byte abgelegt, was den Typ der GOSUB-Anweisung bestimmt. Bei einem normalen GOSUB-Befehl ist dies ein Nullbyte. Handelt es sich jedoch um den Aufruf eines Unterprogramms, die aus einem AFTER oder EVERY-Befehl herrührt, wird zur Unterscheidung eine Eins auf dem Stack abgelegt. Dann folgen die Adresse des nächsten Befehls nach dem GOSUB-Befehl sowie die Adresse der Zeile, in der der GOSUB-Befehl steht. Damit der Stackeintrag beim RETURN-Befehl wieder identifiziert werden kann, wird noch ein Byte auf dem Stack abgelegt, das die Länge des Stackeintrags angibt und damit implizit einen GOSUB-Datensatz kennzeichnet.

Die Daten einer WHILE-WEND-Schleife werden in ähnlicher Form abgelegt.

Lo	**Zeilenadresse des**
Hi	**WHILE-Befehls**
Lo	**Adresse des**
Hi	**WEND-Befehls**
Lo	**Adresse der**
Hi	**WHILE-Bedingung**
&07	**Größe des Stackeintrags**

Der Eintrag enthält also 3 Adressen und als Kennbyte eine 7, die Anzahl der Stackeinträge.

Bei der FOR-NEXT-Schleife wird es etwas komplizierter. Hierbei wird unterschieden, ob die Laufvariable vom Typ Integer oder Real ist. Im ersteren Fall ist nicht nur die Ausführungszeit kürzer, sondern der Stackbedarf ist ebenfalls kleiner. Betrachten wir zuerst den Aufbau bei einer Integerschleife.

Lo	**Adresse der**
Hi	**Laufvariablen**
Lo	**Endwert der**
Hi	**Laufvariablen**
Lo	**STEP-Wert**
Hi	
Sgn	**Vorzeichen des STEP-Werts**
Lo	**Adresse der**
Hi	**FOR-Anweisung**
Lo	**Zeilenadresse der**
Hi	**FOR-Anweisung**
Lo	**Adresse der**
Hi	**NEXT-Anweisung**
Lo	**Zeilenadresse**
Hi	**der NEXT-Anweisung**
&10	**Größe des Stackeintrags**

Der Stackeintrag für eine FOR-NEXT-Scheife mit Integervariable ist also 16 Bytes groß. Wird die Schleife mit einer Real-Variablen durchlaufen, so werden 22 Bytes im Stack belegt.

Lo	**Adresse der**
Hi	**Laufvariablen**
5 Bytes Fließ-kommawert	**Endwert der Laufvariablen**

5 Bytes Fließ-kommawert	**STEP-Wert**
Sgn	**Vorzeichen des STEP-Werts**
Lo **Hi**	**Adresse der** **FOR-Anweisung**
Lo **Hi**	**Zeilenadresse der** **FOR-Anweisung**
Lo **Hi**	**Adresse der** **NEXT-Anweisung**
Lo **Hi**	**Zeilenadresse** **der NEXT-Anweisung**
&16	**Größe des Stackeintrags**

Außer zur Abspeicherung von Schleifenstrukturen wird der BASIC-Stack noch zur Speicherung von Zwischenausdrücken bei numerischen Berechnungen benutzt, z.B. bei verschachtelten Klammerausdrücken und zur Realisierung einer Hierarchie bei den arithmetischen und logischen Operatoren.

3.3 BASIC und Maschinensprache

3.3.1 Der CALL-Befehl

Das Bindeglied zwischen BASIC und Maschinensprache ist der CALL-Befehl. Mit ihm ist es möglich, aus einem BASIC-Programm heraus ein Maschinenprogramm aufzurufen. Zum CALL-Befehl gehört noch eine 16-Bit-Adresse, die besagt, an welcher Stelle das Maschinenprogramm steht, z.B.

```
CALL &8000
```

Damit wird ein Maschinenprogramm ab der Adresse &8000 oder dezimal 32768 aufgerufen. Wird das Maschinenprogramm mit dem RET-Befehl beendet, wird die Kontrolle wieder zurück an den Interpreter gegeben, der in der Ausführung des BASIC-Programms fortfährt.

Beim CALL-Befehl sind Betriebssystem und BASIC-Interpreter nicht direkt zugänglich; im gesamten 64K-Adreßbereich ist RAM selektiert. Selbstverständlich lassen sich jedoch Betriebssystemroutinen über die Einsprungadressen im Bereich ab &B000 aufrufen. Diese Routinen sorgen selbsttätig für die erforderliche ROM/RAM-Konfiguration. Wollen Sie während eines CALL-Befehls auf Routinen des BASIC-Interpreters oder Betriebssystemroutinen zugreifen, die nicht über Vektoren angesprungen werden, so können Sie dazu die RST 3 und RST 5 Routinen benutzen, die die Umschaltung realisieren.

Der CALL-Befehl erlaubt es aber weiterhin noch Parameter von BASIC aus an die Routine zu übergeben. Dazu können hinter der Adresse durch Komma getrennt bis zu 32 Parameter übergeben werden, die der Maschinenroutine dann zur Verfügung stehen. Diese Parameter müssen wie die Adresse selbst einen 16-Bit-Wert ergeben. Sie werden von BASIC auf dem Stack abgelegt. Der BASIC-Interpreter übergibt die Basisadresse dieses Parameterblocks im IX-Register. Im Akku steht zusätzlich noch, wieviele Parameter übergeben wurden. Der letzte Parameter steht also an Adresse IX, der vorletzte

an Adresse IX+2 und der erste Parameter an Adresse IX + 2*(A-1).

Während des CALL-Befehls können sämtliche Registerinhalte verändert werden (bezüglich der Benutzung der Zweitregister siehe im Kapitel Firmware). Auch der Stackpointer darf verändert werden, sofern sichergestellt ist, daß beim abschließenden RET die richtige Rücksprungadresse vom Stack geholt wird.

Bei der Benutzung des CALL-Befehls sind Ihrer Phantasie keine Grenzen gesetzt. Sie können z.B. erweiterte Grafikfunktionen zur Verfügung stellen wie Kreise zeichnen, Fläche ausfüllen usw.

Die Übergabe von Parametern von der Maschinenroutine zurück ans BASIC ist nicht vorgesehen, ist jedoch über einen Umweg möglich. Soll z.B. das Ergebnis eines Maschinenprogramms einer Variablen zugewiesen werden, so kann dem CALL-Befehl deren Adresse übergeben werden. Dies ist mit dem 'Klammeraffen' möglich.

CALL &AB00,@A

Damit steht dem Maschinenprogramm die Adresse der Variablen A zur Verfügung und es kann direkt der Wert der Variablen verändert werden. Diese Möglichkeit ist im Kapitel über den Variablenpointer näher beschrieben.

3.3.2 BASIC-Erweiterungen mit RSX

Betriebssystem und BASIC des CPC unterstützen eine Möglichkeit, eigene Befehle ins BASIC einzubinden, die man RSX nennt. Das ist eine Abkürzung für 'Resident System eXtension'. Diese Erweiterungen können von BASIC aus durch einen Namen aufgerufen werden und erlauben eine Parameterübergabe, wie sie beim CALL-Befehl schon beschrieben wurde. Wenn wir z.B eine Grafikerweiterung

schreiben wollen, die uns ein Quadrat auf den Bildschirm zeichnet, so kann der Aufruf dazu so aussehen:

lQUADRAT,100,100,50

Damit soll ein Quadrat gezeichnet werden, dessen linke obere Ecke die Koordinaten 100,100 hat mit einer Kantenlänge von 50 Punkten.

Wie Sie sehen, wird eine Befehlserweiterung durch einen vorangestellten senkrechten Strich (Shift Klammeraffe) gekennzeichnet.

Eine derartige Befehlserweiterung kann in einem Extensionrom stecken, wie es der Fall ist, wenn Sie ein Diskettenlaufwerk angeschlossen haben oder aber auch im RAM. Damit haben wir die Möglichkeit, eigene Erweiterungen zu schreiben. Damit das Betriebssystem weiß, wo es nach einer solchen Erweiterung suchen soll, muß die Erweiterung erst 'eingebunden' werden. Dazu dient eine Routine des Betriebssystems: KL LOG EXT. Das folgende Beispiel realisiert den oben angesprochenen Befehl zum Zeichnen eines Quadrat und demonstriert, wie die Einbindung vonstatten geht.

```
                    ; RSX-BEFEHLSERWEITERUNG

                    ; LE 15/6/85

BCD1                LOGEXT EQU  &BCD1 ; Erweiterung einbinden
BBC6                ASKCUR EQU  &BBC6 ; Grafik-Cursor holen
BBC0                MOVABS EQU  &BBC0 ; Grafik-Cursor setzen
BBF9                DRAWRE EQU  &BBF9 ; Linie relativ ziehen
BDC7                CHGSGN EQU  &BDC7 ; Vorzeichen ändern

8000                       ORG  &8000

8000  010980               LD   BC,RSX ; Adresse der RSX-Befehlstabelle
```

```
8003  211680             LD   HL,KERNAL ; 4 Byte RAM für Kernal
8006  C3D1BC             JP   LOGEXT ; Erweiterung einbinden

8009  0E80      RSX      DEFW TABLE ; Adresse der Befehlsworte
800B  C31A80             JP   QUADRAT

800E  51554144  TABLE    DEFM "QUADRA"
8014  D4                 DEFB "T"+&80
8015  00                 DEFB 0 ; Ende der Tabelle

8016            KERNAL DEFS 4 ; Speicher für Kernal

801A  FE03      QUADRA CP   3 ; drei Parameter ?
801C  C0                 RET  NZ
801D  CDC6BB             CALL ASKCURS ; Grafik-Cursor holen
8020  D5                 PUSH DE ; X-Koordinate merken
8021  E5                 PUSH HL ; Y-Koordinate merken
8022  DD5605             LD   D,(IX+5)
8025  DD5E04             LD   E,(IX+4) ; X-Koordinate
8028  DD6603             LD   H,(IX+3)
802B  DD6E02             LD   L,(IX+2) ; Y-Koordinate
802E  CDC0BB             CALL MOVABS ; Grafik-Cursor auf X,Y-Koordinate
8031  DD5601             LD   D,(IX+1)
8034  DD5E00             LD   E,(IX) ; Länge nach de als X-Offset
8037  D5                 PUSH DE ; merken
8038  210000             LD   HL,0 ; Y-Offset
803B  CDF9BB             CALL DRAWREL ; waagerechte Linie ziehen
803E  E1                 POP  HL
803F  E5                 PUSH HL
8040  CDC7BD             CALL CHGSGN ; Y-Offset negativ
8043  E5                 PUSH HL
8044  110000             LD   DE,0
8047  CDF9BB             CALL DRAWREL ; senkrechte Linie ziehen
804A  D1                 POP  DE ; negativer X-Offset
804B  210000             LD   HL,0 ; Y-Offset null
804E  CDF9BB             CALL DRAWREL ; waagerechte Linie ziehen
8051  E1                 POP  HL
8052  110000             LD   DE,0
8055  CDF9BB             CALL DRAWREL ; senkrechte Linie ziehen
```

```
8058  E1          POP  HL
8059  D1          POP  DE
805A  C3C0BB      JP   MOVABS ; Koordinaten wiederherstellen
```

Nachdem das Programm geladen (als Binärfile von Diskette) oder mittels DATA-Lader generiert im Speicher steht, muß es einmalig initialisiert werden werden. Dies geschieht durch Aufruf von CALL &8000. Damit steht der neue Befehl zur Verfügung. Für die Einbindung werden zwei Tabellen benutzt. Die erste, in unserem Beispiel RSX genannt, enthält zuerst die Adresse der zweiten Tabelle, hier TABLE genannt, und anschließend Sprungbefehle auf die eigentliche Erweiterung. Die zweite Tabelle enthält die Namen, unter denen die neuen Befehle aufgerufen werden können. Dabei sind Großbuchstaben sowie Punkte erlaubt. Das letzte Zeichen eines Befehlswort wird durch das gesetzte Bit 7 gekennzeichnet. Danach können weitere Befehlsworte folgen. Das Ende der Tabelle ist durch ein Nullbyte gekennzeichnet. In jeder Tabelle müssen natürlich gleich viel Einträge stehen; zu jedem Befehlswort muß in der ersten die korrespondierende Sprungadresse stehen. Unter dem Label KERNAL müssen wir dem Betriebssystem 4 Bytes zur Verfügung stellen, die zur Verwaltung der Erweiterung benutzt werden. Die 4 Bytes müssen zwischen Adresse &4000 und &BFFF stehen.

Die Routine zum Zeichnen des Quadrats beginnt beim Label QUADRAT. Zuerst wird geprüft, ob drei Parameter übergeben wurden. Ist das nicht der Fall, wird sofort zurückgekehrt. Andernfalls wird die aktuelle Grafikcursorposition geholt und auf den Stack gerettet. Jetzt werden die übergebenen X- und Y-Koordinaten nach DE und HL geholt. Die Basis des Parameterblock steht in IX zur Verfügung. Nachdem der Grafikcursor auf diese Koordinaten gesetzt wird, kann viermal die Routine zum Zeichnen einer Linie relativ zur augenblicklichen Position aufgerufen werden. Um einen negativen Offset zu berechnen, wird die Routine CHGSGN der Integerarithmetik aufgerufen. Zum Abschluß wird die ursprüngliche Grafikcursorposition wieder hergestellt.

Als Beispiel für eine Anwendung der Routine geben Sie bitte folgendes kleine Programm ein:

```
10 CLS
20 FOR i=35 TO 400 STEP 20
30 lQUADRAT,i,i,30
40 NEXT
```

3.3.3 Der Variablenpointer '@'

Eine besonders für den Maschinenprogrammierer interessante Funktion ist der Variablenpointer, der durch den 'Klammeraffen' aufgerufen wird. Diese Funktion gibt die Adresse zurück, an der eine Variable im Speicher abgelegt ist. Der Aufruf sieht so aus:

```
PRINT @a
```

Damit wird die Adresse der Variablen a ausgegeben. War die Variable noch nicht angelegt, wird die Fehlermeldung 'Improper argument' ausgegeben.

Wenn wir nun an den Inhalt der Variablen wollen, müssen wir zwischen den 3 möglichen Typen unterscheiden.

Bei Integervariablen sieht die Sache am einfachsten aus. An der angegebenen Adresse ist der 16-Bit-Wert abgelegt. Den Wert der Variablen a% erhalten wir also mit folgender Formel:

```
PRINT PEEK(@a%)+256*PEEK(@a%+1)
```

Dabei können wir Werte zwischen 0 und 65535 erhalten. Soll das Vorzeichen noch berücksichtigt werden, müssen wir die Funktion UNT anwenden.

```
PRINT UNT(PEEK(@a%)+256*PEEK(@a%+1))
```

Bei Fließkommavariablen zeigt der Variablenpointer ebenfalls

auf den Wert der Variablen, der sich jedoch aus 5 Bytes zusammensetzt. Die ersten 4 Bytes sind die sogenannte Mantisse und das 5. Byte ist der Zweierexponent, mit dem die Mantisse multipliziert werden muß, um den Wert der Variablen zu erhalten. Wenn wir die 4 Mantissenbytes mit m1 bis m4 bezeichnen und den Exponenten mit ex, so ergibt folgende Formel den zugehörigen Fließkommawert:

```
x = (1-2*SGN (m4 AND 128)) * 2 ^ (ex-129) *
    (1+ ((m4 AND 127)+(m3+(m2+m1/256)/256)/256)/128)
```

Aus der Formel wird deutlich, daß das Vorzeichen der Fließkommazahl im obersten Bit von m4 steckt und daß die Mantissenbytes m1 bis m4 steigende Wertigkeiten haben. Der Zweierexponent enthält einen Offset von 129, so daß sich Werte von 2^-129 bis 2^127 ergeben. Probieren wir unsere Formel einmal aus.

```
100 a=-13:' untersuchte Fließkommavariable
110 ad = @a: ' Adresse von a
120 m1=PEEK(ad):m2=PEEK(ad+1):m3=PEEK(ad+2)
130 m4=PEEK(ad+3):ex=PEEK(ad+4)
140 PRINT (1-2*SGN (m4 AND 128)) * 2 ^ (ex-129) *
    (1+ ((m4 AND 127)+(m3+(m2+m1/256)/256)/256)/128)
```

Wenn Sie das Programm laufen lassen, erhalten Sie den Wert -13 zurück. Ersetzen Sie Zeile 100 einmal durch INPUT a, so können Sie beliebige Werte ausprobieren.

Anwendung findet die Variablenpointer-Funktion im CALL-Befehl, der ja bekanntlich nur 16-Bit-Werte übergeben kann. Wollen Sie also mit Fließkommawerten arbeiten, so können Sie mit '@' die Adresse einer Fließkommazahl übergeben, auf die Sie sich dann beziehen können. Mit dieser Methode ist es natürlich auch möglich, den Wert der Fließkommavariablen direkt zu verändern.

Noch interessanter wird es bei Stringvariablen. Auch hier können wir den Variablenpointer benutzen, der uns wieder die

Adresse der Variablen zurückgibt. Dies ist jedoch nicht direkt die Adresse des Strings, sondern des sogenannten Stringdescriptors. Dieser Stringdescriptor ist drei Bytes lang. Das erste Byte enthält die Länge des Strings, also einen Wert von Null bis 255. Die nächsten beiden Bytes enthalten die Adresse des Strings.

```
100 INPUT a$
110 ad=@a$
120 l=PEEK(ad)
130 sa=PEEK(ad+1)+256*PEEK(ad+2)
140 FOR i=sa TO sa+l-1: PRINT CHR$(PEEK(I));:NEXT
```

Das Programm holt Länge und Adresse des Strings, liest und gibt ihn aus.

Auch hier kann über den Variablenpointer ein String an den CALL-Befehl übergeben werden.

Strings lassen sich im Zusammenhang mit dem CALL-Befehl noch ganz anders einsetzen. Man legt dazu ein Maschinenprogramm einfach in einem String ab und kann es mit dem CALL-Befehl und dem Variablenpointer aufrufen. Das Maschinenprogramm muß dazu verschiebbar (keine internen absoluten Sprünge) und darf nicht länger als 255 Bytes sein. Das ist bei kleineren Utilities meist gegegeben. Wollen Sie diese Methode anwenden, so müssen Sie folgendermaßen vorgehen:

Zuerst wird das Maschinenprogramm in die Stringvariable abgelegt. Dies wird meist mit READ und DATA geschehen. Wollen Sie das Programm dann ausführen, so berechnen Sie die Startadresse des Strings (und des Maschinenprogramms) mit dem 'Klammeraffen'.

3.4 Das BASIC-ROM

3.4.1 Die Fließkommaarithmetik

Sämtliche arithmetische Funktionen, die der BASIC-Interpreter benutzt, stehen im Betriebssystem-ROM. Sie werden über die Sprungtabelle von &BD5E bis &BDBB aufgerufen. Wenn Sie diese Arithmetikroutinen ändern wollen, so brauchen Sie nur an der entsprechenden Stelle einen Sprung auf Ihre Routine einfügen.

Als ein Beispiel zur Anwendung der Fließkommaroutinen des BASIC-Interpreters zeigen wir Ihnen eine Routine zur Berechnung der Quadratwurzel einer Zahl. Der BASIC-Interpreter des CPC 664 & 6128 stellt uns diese Funktion zwar schon zur Verfügung, jedoch soll gezeigt werden, daß durch Anwendung von leistungsfähigen Algorithmen diese noch verbessert werden können.

Die eingebaute SQR-Funktion arbeitet nach dem gleichen Algorithmus, nach dem auch die Potenzberechnung geschieht.

SQR(X) = EXP (LOG(X)*0.5)

Es müssen also jedesmal Exponential- und Logarithmusfunktion berechnet werden, was über aufwendige Polynomberechnungen geschieht. Die Quadratwurzel läßt sich jedoch über eine einfache Iterationsvorschrift berechnen.

X(N+1) = (X(N) + A / X(N)) / 2

wobei A die Zahl ist, aus der die Wurzel gezogen werden soll und X(N) der alte und X(N+1) der neue Näherungswert ist. Als Startwert kann man die Zahl A selbst nehmen. Ein besserer Nährungswert ergibt sich jedoch, wenn man den Zweierexponent der Fließkommazahl halbiert. Dann ändert sich das Ergebnis nach vier Iterationen im Rahmen der Rechengenauigkeit nicht mehr. Beachten Sie auch, daß die Division durch 2 nicht als aufwendige Fließkommadivision realisiert wurde, sondern

einfach dadurch, daß man den Zweierexponenten um eins erniedrigt. Der Zeitgewinn dieses Verfahrens ist bedeutend. Benötigt die SQR-Routine des Interpreters noch 27 Millisekunden, so schafft unsere Routine die gleiche Aufgabe in knapp 8 Millisekunden; sie ist also mehr als dreimal so schnell.

```
                    ;SCHNELLE SQR-ROUTINE
                    ;LE 10/6/85

A000                        ORG  &A000

BD91                SGN     EQU  &BD91
BD85                DIV     EQU  &BD85
BD79                ADD     EQU  &BD79

A000  CD91BD        NEWSQR  CALL SGN ; Vorzeichen prüfen
A003  3F                    CCF
A004  C8                    RET  Z ; Null, schon fertig
A005  F20CA0                JP   P,GOON
A008  3E01                  LD   A,1 ; 'IMPROPER ARGUMENT'
A00A  B7                    OR   A
A00B  C9                    RET

A00C  E5            GOON    PUSH HL
A00D  1153A0                LD   DE,STORE1
A010  010500                LD   BC,5
A013  EDB0                  LDIR ; Radikant merken
A015  E1                    POP  HL

A016  E5                    PUSH HL
A017  DDE1                  POP  IX
A019  DD7E04                LD   A,(IX+4) ; Exponent
A01C  D681                  SUB  &81 ; normalisieren
A01E  3F                    CCF
A01F  1F                    RRA  ; Exponent halbieren
A020  C601                  ADD  A,1
A022  DD7704                LD   (IX+4),A ; als Startwert
```

```
A025  0604            LD   B,4 ; 4 Iterationen
A027  C5       ITER   PUSH BC
A028  E5              PUSH HL
A029  1158A0          LD   DE,STORE2
A02C  010500          LD   BC,5
A02F  EDB0            LDIR ; Näherung merken
A031  E1              POP  HL
A032  E5              PUSH HL
A033  1153A0          LD   DE,STORE1
A036  EB              EX   DE,HL
A037  010500          LD   BC,5
A03A  EDB0            LDIR ; Radikant holen
A03C  E1              POP  HL
A03D  1158A0          LD   DE,STORE2
A040  CD85BD          CALL DIV
A043  1158A0          LD   DE,STORE2
A046  CD79BD          CALL ADD
A049  E5              PUSH HL
A04A  DDE1            POP  IX
A04C  DD3504          DEC  (IX+4) ; ZAHL / 2
A04F  C1              POP  BC
A050  10D5            DJNZ ITER
A052  C9              RET

A053           STORE1 DEFS 5
A058           STORE2 DEFS 5
```

Wie kann man aber den Interpreter dazu veranlasssen, die neuen Routine zu verwenden? Für die SQR-Funktion dient der Vektor &BD9D. An diese Adresse muß ein Sprung auf unsere Routine eingetragen werden.

JP &A000

Wenn die Routine von BASIC aus aufgerufen wird, muß das HL-Register auf den Fließkommawert zeigen. Nach Ausführung der Routine muß das HL-Register auf das Ergebnis zeigen. Normalerweise hat sich der Wert dieses Registers nicht verändert. Die Flags zeigen den Fehlerstatus der Funktion an:

Besitzen Sie einen CPC 6128, so müssen Sie die Adressen der Routinen SGN, DIV und ADD um jeweils drei erhöhen; ebenso muß der Sprung nach &A000 dann an Adresse &BDA0 stehen.

Fehlerstatus der Funktionen

C=1	**ordnungsgemäße Ausführung**
C=0 & Z=1	**'Division by zero'**
C=0 & N=1	**'Overflow'**
C=0 & Z=0	**'Improper argument'**

Auf den nächsten Seiten finden Sie das Listing der Fließkommaarithmetik, wobei jede Routine auch die Adresse der Sprungtabelle enthält, über die sie vom BASIC-Interpreter aus angesprochen wird. Die Integerarithmetik befindet sich im BASIC-ROM von Adresse DD2F bis DE19. Sie wird von Interpreter immer dann benutzt, wenn dies möglich ist. Beim CPC 464 befand sich auch die Integerarithmetik im unteren ROM und wurde von BASIC über Vektoren aufgerufen. Da bei der Integerrechnung immer nur mit 2-Byte-Werten gearbeitet wird, ist diese Arithmetik bedeutend schneller als die Rechnung mit Fließkommazahlen. Nutzen Sie dies auch in Ihren Programmen aus und verwenden Sie wenn immer möglich nur Integervariablen. Dies gilt besonders auch für FOR-NEXT-Schleifen (siehe dazu auch Kapitel 3.2).

```
******  BD97  PI
2F73
2F78    PI
******  BD5E  Variable kopieren
2F91
******  BD64  4-Byte-Wert nach Fließkomma
2FC8
******  BDB5  4-Byte-Wert mal 256 nach Integer
2FD1
******  BD67  Fließkomma nach Integer
2FD9
******  BD6A  Fließkomma nach Integer
3001
******  BD6D  FIX
3014
******  BD70  INT
3055
******  BD73
305F
*****   BD76  Zahl mit 10^A multiplizieren
30C6
*****   BDB8  RND INIT
3136
*****   BDBB  SET RANDOM SEED
3143
*****   BD7C  RND
3159
*****   BD88  letzten RND-Wert holen
3188
******  BDA3  LOG10
31B1
******  BDA0  LOG
31B6
******  BDA6  EXP
322F
******  BD9A  SQR
32AC
******  BD9D  Potenzierung
```

32AF
****** BD94 DEG/RAD
3345
****** BDAA COS
3349
****** BDA7 SIN
3353
****** BDAF TAN
33C8
****** BDB2 ATN
33D8
****** BD7F Subtraktion
349E
****** BD79 Addition
34A2
****** BD82 Multiplikation
3577
****** BD85 Division
3604
****** BD8B Vergleich
36DF
****** BD91 SGN
3727
****** BD8E Vorzeichenwechsel
3731

******	CPC 664 & 6128 BASIC 1.1
C000	erstes Vordergrund-ROM
C001	Mark 1
C002	Version 1
C003	Modifikation 0
C004	Adresse des Namens
******	BASIC-Initialisierung
C006	Stack ab C000
C009	KL ROM WALK
C00C	Speicher konfigurieren
C00F	zuwenig Speicher, dann Reset
C013	Flag für Blanks unterdrücken löschen
C016	Zeiger auf ' BASIC 1.1'
C019	Text ausgeben
C01C	aktuelle Zeilenadresse auf Null
C01F	Fehlernummer löschen
C022	RND-Init
C025	AUTO-Modus löschen
C028	NEW-Befehl
C02B	240
C02E	SYMBOL AFTER 240
C031	zum READY-Modus
C033	' BASIC 1.1', LF,LF,0
C040	'BASI', 'C'+80H,0
*******	BASIC-Befehl EDIT
C046	Zeilennummer nach DE holen
C04A	Stack initialisieren
C04D	BASIC-Zeile DE suchen
C050	BASIC-Zeile in Puffer listen
C053	Eingabezeile holen
*******	READY-Modus
C058	Stack initialisieren
C05B	diverse Initialisierungen
C05E	Zeilenadresse holen
C061	SOUND HOLD
C064	Break-Event löschen
C067	Bildschirm initialisieren
C06A	geschütztes Programm ?

C06E	ja, Programm und Variablen löschen
C071	ERROR-Nummer
C074	'Syntax error' ?
C076	nein
C078	ERROR-Nummer auf Null
C07B	Zeilennummer der ERROR-Zeile holen
C07F	zum EDIT-Befehl
C081	Zeiger auf 'Ready'
C084	ausgeben
C087	aktuelle Zeilenadresse auf Null
C08A	AUTO-Flag gesetzt ?
C08E	nein
C090	nächste Zeilennummer vorgeben
C093	zum READY-Modus
C095	Blank, TAB und LF überlesen
C09D	Blank, TAB und LF überlesen
C0AF	Eingabezeile holen
C0B2	'ESC' gedrückt, dann wiederholen
C0B4	LF ausgeben
C0B7	Blank, TAB und LF überlesen
C0D4	zur Interpreterschleife
C0D7	'Ready', LF,0
*******	AUTO-Modus löschen
C0DF	
*******	AUTO-Modus setzen
C0E1	Zeilennummer
C0E6	Flag für AUTO setzen
*******	BASIC-Befehl AUTO
C0EA	10, Default
C0EF	','
C0F1	Zeilennummer nach DE holen
C0F5	10, Default
C0F8	folgt Komma ?
C0FB	ja, Zeilennummer nach DE holen
C0FE	Zeilenende, sonst 'Syntax error'
C102	AUTO-Inkrement merken
C106	Flag für AUTO-Modus merken
C10D	Zeilennummer
C115	AUTO-Modus löschen

C118	Zeile editieren
C11E	Zeilennummer
C121	plus Inkrement
C122	AUTO-Modus setzen
******	BASIC-Befehl NEW
C128	
C129	Programm und Variablen löschen
C12C	zum READY-Modus
******	BASIC-Befehl CLEAR
C12F	'INPUT'
C13A	diverse Initialisierungen
******	CLEAR INPUT
C13F	Blanks überlesen
******	Programm und Variablen löschen
C145	Beginn des freien RAMs
C149	HIMEM
C152	Akku löschen
C154	Beginn freies RAM bis HIMEM löschen
C156	Flag für geschütztes Programm löschen
C159	Variablenzeiger rücksetzen
C15F	Disk I/O abbrechen
C163	RAD-Modus setzen
C166	Descriptorstack initialisieren
C16C	Stream-Reset
C16F	TROFF
C172	AUTO-Modus löschen
C175	diverse Initialisierungen, s.u.
C17A	Strings löschen
C17D	Variablenzeiger rücksetzen
C180	alle Variablen auf Typ 'Real'
C183	Blank, TAB und LF überlesen
******	diverse Initialisierungen
C189	Tabulator-Stops initialisieren
C18C	Programmzeiger löschen
C18F	ON ERROR löschen
C192	Programmzeiger nach Unterbrechung löschen
C195	SOUND und Event-Reset
C198	BASIC-Stack initialisieren
C19B	Flag für FN löschen

C19E	RESTORE
C1AB	< 8 ?
C1AD	TXT STR SELECT
C1B1	aktuelle Streamnummer
C1B7	Eingabekanal
C1C1	aktuelle Streamnummer
C1C4	Drucker ?
C1C7	Eingabekanal
C1CA	Diskette ?
C1CD	auf Streamnummer testen
C1D2	auf Streamnummer testen
C1D7	auf Streamnummer testen
******	Streamnummer holen
C1E8	auf Streamnummer testen
C1ED	'Improper argument'
C1F5	','
C1F7	Sprung nach (BC), Funktion ausführen
******	auf Streamnummer testen
C1FF	'#'
C201	0 als Default
C204	Streamnummer holen
C208	folgt Komma ?
C20B	nein, dann Ende des Statements
******	Streamnummer holen
C210	Test auf nachfolgendes Zeichen
C213	"#"
C214	10, Maximalwert+1
C218	Maximalwert nach B
C219	8-Bit-Wert holen
C21C	mit Maximalwert vergleichen
C21F	kleiner, ok
C220	'Improper argument'
******	8-Bit-Wert kleiner 2 holen
C223	Maximalwert 2
C225	Argument holen und testen
******	BASIC-Befehl PEN
C227	Streamnummer holen
C22A	TXT SET PEN
C230	folgt Komma ?

C234	8-Bit-Wert kleiner 2 holen
C237	TXT SET BACK
******	BASIC-Befehl PAPER
C23C	Streamnummer holen
C23F	TXT SET PAPER
C242	Argument < 16 holen
C246	Sprung nach (BC), Funktion ausführen
******	BASIC-Befehl BORDER
C24B	2 Argumente kleiner 32 holen
C24F	SCR SET BORDER
******	BASIC-Befehl INK
C254	Argument kleiner 16 holen
C258	Test auf ','
C25B	2 Argumente kleiner 32 holen
C260	SCR SET INK
******	2 Argumente kleiner 32 holen
C265	Argument < 32 holen
C268	nach B
C269	folgt Komma ?
C26D	32
C26F	Argument kleiner 32 holen
C272	nach C
******	Argument < 16 holen
C274	16
C276	Argument kleiner 16 holen
******	BASIC-Befehl MODE
C278	3
C27A	Argument kleiner 3 holen
C27E	SCR SET MODE
******	BASIC-Befehl CLS
C283	Streamnummer holen
C287	TXT CLEAR WINDOW
C28C	Streamnummer holen
C291	'Improper argument'
C294	Test auf ')'
******	BASIC-Funktion COPYCHR$
C29B	Streamnummer holen, Klammer zu
C29E	TXT RD CHAR
C2A1	Zeichen als String übernehmen

******	BASIC-Funktion VPOS
C2A4	Streamnummer holen
C2A8	Cursorzeile holen
******	BASIC-Funktion POS
C2AD	Streamnummer holen
C2B0	Test auf ')'
C2B4	Position holen
C2B7	Akkuinhalt als Integerzahl übernehmen
******	aktuelle PRINT-Position holen
******	Cursorzeile holen
C2CA	TXT GET CURSOR
C2CD	TXT VALIDATE
C2DA	TXT GET WINDOW
******	BASIC-Befehl LOCATE
C302	Streamnummer holen
C305	zwei 8-Bit-Werte ungleich Null holen
C30C	TXT SET CURSOR
******	BASIC-Befehl WINDOW
C311	'SWAP'
C315	Streamnummer holen
C318	zwei 8-Bit-Werte ungleich Null holen
C31C	Test auf ','
C31F	zwei 8-Bit-Werte ungleich Null holen
C326	TXT WIN ENABLE
******	WINDOW SWAP
C32B	Blanks überlesen
C32E	Argument < 8 holen
C332	folgt Komma ?
C335	Default Null
C337	ja, Argument < 8 holen
C33C	TXT SWAP STREAMS
******	Argument < 8 holen
C341	8, Maximalwert
C343	Argument holen
******	BASIC-Befehl TAG
C346	Streamnummer holen
******	BASIC-Befehl TAGOFF
C34D	Streamnummer holen
C351	TXT SET GRAPHIC

******	zwei 8-Bit-Werte ungleich Null holen
C354	ersten Wert holen
C357	nach D
C358	Test auf ','
C35C	8-Bit-Wert ungleich Null holen
C360	Wert nach E
******	BASIC-Befehl CURSOR
C363	Streamnummer holen
C366	Null ?
C368	8-Bit-Wert < 2 holen
C36C	TXT CUR OFF
C36F	TXT CUR ON
C372	folgt Komma ?
C375	nein
C376	8-Bit-Wert < 2 holen
C37A	TXT CUR DISABLE
C37D	TXT CUR ENABLE
******	String ausgeben
C380	Stringadresse
C381	132
C384	WIDTH auf 132
C387	POS auf Eins setzen
C390	Zeichen holen
C391	Zeiger erhöhen
C392	letztes Zeichen ?
C393	nein, ausgeben
C396	nächstes Zeichen
******	LF ausgeben
C39C	LF
C39E	ausgeben
******	Zeichen ausgeben
C3A5	Zeichen ausgeben
C3AB	LF
C3AD	nein
C3AF	Ausgabegerät
C3B2	Drucker ?
C3B5	Disk ?
C3B8	Zeichen ausgeben
******	Zeichen ausgeben

C3BE	Zeichen ausgeben
******	Ausgabekanal selektieren
C3C4	Ausgabekanal
C3C9	auf Drucker ausgeben
C3CC	auf Disk
C3D0	auf Bildschirm
C3D4	TXT SET GRAPHIC
C3D9	TXT SET BACK
C3DD	TXT VDU ENABLE
C3E0	TXT VALIDATE
C3E5	CR
C3EA	LF
C3EC	TXT OUTPUT
C3F1	TXT VALIDATE
******	CR & LF auf Drucker ausgeben
C3F8	CR
C3FD	LF
C40B	MC PRINT CHAR
C40F	Test auf Abbruch mit 'ESC'
C415	CR
C41A	' '
C434	CR
C439	LF
C449	DISK OUT CHAR
C44C	fehlerfrei ?
C44D	Fehlermeldung ausgeben
C44F	DERR setzen
C453	CAS TEST EOF
C45A	Vorzeichen als Integerzahl übernehmen
C45F	CAS IN CHAR
C462	fehlerfrei ?
C468	Fehlermeldung ausgeben
C46B	'Diskettenfehler'
******	POS auf Eins setzen
C472	KM READ CHAR
******	Test auf Abbruch mit 'ESC'
C475	KM READ CHAR
C479	'Break'
C47C	auf zweiten Tastendruck warten

C47F	'ESC', dann Abbruch
C485	Adresse der Break-Event-Routine
C488	BASIC-ROM-Select
C48E	KM ARM BREAK
******	Break-Event-Routine
C495	KM READ CHAR
C498	keine Taste gedrückt ?
C49A	Break durch 'ESC'
C49C	Tastendrücke vor 'ESC' ignorieren
C49E	warten auf zweites 'ESC'
C4A1	Test auf ON BREAK GOSUB
******	Warten auf Tastendruck nach 'ESC'
C4A7	SOUND HOLD
C4AF	TXT CUR ON
C4B2	KM WAIT CHAR
C4B5	'ESC'
C4BB	TXT CUR OFF
C4C3	' '
C4C5	KM CHAR RETURN
C4CA	SOUND CONTINUE
C4DC	KM DISARM BREAK
******	BASIC-Befehl ORIGIN
C4E1	2 Argumente holen
C4E6	folgt Komma ?
C4E9	nein
C4EB	2 Argumente holen
C4F0	Test auf ','
C4F3	2 Argumente holen
C4F8	GRA WIN HEIGHT
C4FE	GRA WIN WIDTH
C504	GRA SET ORIGIN
C509	Ende des Statements ?
C510	GRA CLEAR WINDOW
******	BASIC-Befehl FILL
C515	Argument < 16 holen
C51A	Garbage Collection
C51D	freien Speicherplatz berechnen
C520	mindestens 29 Bytes
C523	Vergleich HL <> BC

C526	sonst 'Memory full'
C528	Fehlermeldung ausgeben
C52D	FILL
******	BASIC-Befehl MOVE
C532	GRA MOVE ABSOLUTE
******	BASIC-Befehl MOVER
C537	GRA MOVE RELATIVE
******	BASIC-Befehl DRAW
C53C	GRA LINE ABSOLUTE
******	BASIC-Befehl DRAWR
C541	GRA LINE RELATIVE
******	BASIC-Befehl PLOT
C546	GRA PLOT ABSOLUTE
******	BASIC-Befehl PLOTR
C54B	GRA PLOT ABSOLUTE
C54F	2 Integerargumente holen
C552	folgt Komma ?
C555	nein
C557	','
C55C	folgt Komma ?
C55F	nein
C563	8-Bit-Wert < 4 holen
C567	SCR ACCESS
C56F	Sprung nach (BC)
******	BASIC-Funktion TEST
C574	GRA TEST ABSOLUTE
******	BASIC-Funktion TESTR
C579	GRA TEST RELATIVE
C57D	2 Argumente holen
C580	Test auf ')'
C587	Sprung nach (BC)
C58A	Akkuinhalt als Integerzahl übernehmen

******	2 Integerargumente holen
C58F	16-Bit-Wert -32768 bis 32767 holen
C593	Test auf ','
C596	16-Bit-Wert -32768 bis 32767 holen
C59A	Ergebnis nach BC
******	BASIC-Befehl GRAPHICS
C59D	'PAPER'
C5A1	Test auf nachfolgendes Zeichen
C5A4	'PEN'
C5A5	','
C5AA	folgt Komma ?
C5AE	8-Bit-Wert < 2 holen
******	GRAPHICS PAPER
C5B4	Blanks überlesen
C5B7	Argument < 16 holen
C5BA	GRA SET PAPER
******	GRAPHICS PEN
C5BD	Argument < 16 holen
C5C0	GRA SET PEN
******	BASIC-Befehl MASK
C5C3	','
C5C7	8-Bit-Argument holen
C5CD	folgt Komma ?
C5D1	8-Bit-Wert < 2 holen
******	BASIC-Befehl FOR
C5D7	Variable lesen
C5DD	zugehöriges NEXT suchen
C5E0	Adresse merken
C5E6	offene FOR-NEXT-Schleife suchen
C5E9	gefunden, BASIC-Stackpointer setzen
C5ED	Ende des Statements ?
C5F0	Default Null
C5F3	nein, Variable holen
C5FC	Vergleich HL<>DE

C5FF	'Unexpected NEXT'
C603	aktuelle Zeilenadresse nach HL
C607	aktuelle Zeilenadresse setzen
C610	22 Bytes, Typ 5 'Real'
C616	16 Bytes, Typ 2 'Integer'
C61A	'Type mismatch'
C61C	Fehlermeldung ausgeben
C61F	Anzahl Bytes nach A
C620	Platz im BASIC-Stack reservieren
C624	Variablenadresse auf BASIC-Stack
C628	Test auf '='
C62B	Ausdruck holen
C62F	Variablentyp vergleichen
C633	Zwischenspeicher für FOR-Variable
C636	Variable nach HL kopieren
C63A	Test auf nachfolgendes Zeichen
C63D	'TO'
C63E	Ausdruck holen
C643	Variablentyp vergleichen
C646	Endwert auf BASIC-Stack
C64C	eins als Default STEP-Wert
C64F	Integerzahl HL übernehmen
C653	nächstes Zeichen
C654	'STEP' ?
C656	nein
C658	Blanks überlesen
C65B	Ausdruck holen
C65F	Variablentyp vergleichen
C663	Variable nach (HL) kopieren

C666	Vorzeichen holen
C66A	Vorzeichen von STEP-Wert auf BASIC-Stack
C66E	Ende des Statements, sonst 'Syntax error'
C673	Adresse des FOR-Befehls auf BASIC-Stack
C677	aktuelle Zeilenadresse nach HL
C67C	Zeilenadresse von FOR auf BASIC-Stack
C681	Adresse des NEXT-Befehls auf BASIC-Stack
C689	Zeilenadresse NEXT-Befehl auf BASIC-Stack
C68C	#10 oder #16 für Integer/Real auf Stack
C68E	Zeiger auf Zwischenspeicher
C691	FOR-Variable zurückholen
C695	Flag für ersten Durchlauf
C699	aktuelle Zeilenadresse setzen
C69F	zum NEXT-Befehl
C6A1	Fehlermeldung ausgeben
C6A4	'Unexpected NEXT'
*******	BASIC-Befehl NEXT
C6A5	
C6A7	Flag für Inkrement addieren
C6AB	offene FOR-NEXT-Schleife suchen
C6B1	BASIC-Stackpointer setzen
C6B6	Test auf Schleifenende
C6BE	Programmzeiger nach DE
C6C2	Zeilenadresse nach HL
C6C5	aktuelle Zeilenadresse setzen
C6CA	BASIC-Stackpointer
C6CD	plus 5
C6CF	Programmzeiger nach 'NEXT'
C6D2	BASIC-Stackpointer setzen
C6D6	folgt Komma ?
C6D9	ja, nächste NEXT-Schleife

******	offene FOR-NEXT-Schleife suchen
C6DC	BASIC-Stackpointer
C6EB	'WHILE-WEND' ?
C6F8	Vergleich HL <> DE
C70A	Integer ?
C70C	ja
C713	Variablentyp und -Adresse setzen
C717	Flag für ersten Durchlauf
C71B	ja, Addition überspringen
C71F	Addition
C729	Variable nach (HL) kopieren
C730	arithmetischer Vergleich
C734	10
C73E	erster Durchlauf ?
C742	ja, Addition überspringen
C749	STEP-Wert nach HL holen
C74C	Integer-Addition HL := HL + DE
C74F	'Overflow'
C751	Fehlermeldung ausgeben
C761	Integer-Vergleich
******	BASIC-Befehl IF
C76A	Ausdruck holen
C76D	'GOTO'
C771	Test auf nachfolgendes Zeichen
C774	'THEN'
C778	Ende der Zeile oder ELSE-Zweig suchen
C77C	Ende des Statements ?
C77F	ja
C780	Zeilennummer
C782	ja, zum GOTO-Befehl
C784	Zeilenadresse ?
C786	nein, BASIC-Befehl ausführen
******	BASIC-Befehl GOTO
C789	Zeilenadresse holen
C78D	Adresse als Programmzeiger übernehmen

******	BASIC-Befehl GOSUB
C78F	Zeilenadresse holen
C794	Kennzeichen für normales 'GOSUB'
C796	Adresse des Unterprogramms merken
C797	6 Bytes
C799	Platz im BASIC-Stack reservieren
C79F	Adresse der Anweisung nach 'GOSUB'
C7A0	auf BASIC-Stack
C7A3	aktuelle Zeilenadresse nach HL
C7A8	Zeilenadresse auf BASIC-Stack
C7AB	Kennzeichen für 'GOSUB'
C7B1.	Programmzeiger auf Unterprogramm
*******	BASIC-Befehl RETURN
C7B4	'GOSUB' auf BASIC-Stack suchen
C7B7	BASIC-Stackpointer zurücksetzen
C7BA	Kennbyte
C7BD	Adresse der Anweisung nach 'GOSUB'
C7BE	nach DE holen
C7C1	Zeilenadresse nach HL
C7C4	aktuelle Zeilennummer setzen
C7C8	Kennbyte
C7C9	kleiner eins ?
C7CB	ja, normales 'GOSUB'
C7CC	eins, dann GOSUB nach AFTER/EVERY
C7CF	zur Event-Routine
C7D6	Kennzeichen von BASIC-Stack holen
C7DB	BASIC-Stackpointer zurücksetzen
C7E0	'GOSUB'
C7E6	Fehlermeldung ausgeben
C7E9	'Unexpected RETURN'
******	BASIC-Befehl WHILE
C7EB	zugeghöriges WEND suchen
C7EE	Adresse merken
C7F0	Zeilenadresse für 'WHILE-WEND'

C7F6	BASIC-Stackpointer setzen
C7F9	7 Bytes
C7FB	Platz im BASIC-Stack reservieren
C7FF	aktuelle Zeilenadresse nach HL
C804	Zeilenadresse auf BASIC-Stack
C809	Adresse nach 'WEND' auf BASIC-Stack
C810	Adresse der WHILE-Bedingung
C811	auf BASIC-Stack
C813	Kennzeichen für 'WHILE'
C816	BASIC-Stackpointer setzen
C81B	WHILE-Bedingung testen
******	BASIC-Befehl WEND
C81D	
C822	'Unexpected WEND'
C824	Fehlermeldung ausgeben
C82C	BASIC-Stackpointer setzen
C82F	aktuelle Zeilenadresse nach HL
C832	Zeilenadresse für WHILE-WEND
C83B	aktuelle Zeilenadresse setzen
C848	Ausdruck holen
C84B	Vorzeichen holen
C84F	Bedingung erfüllt ?
C850	Zeilenadresse für WHILE-WEND
C853	als aktuelle Zeilenadresse setzen
C858	Platz im BASIC-Stack freigeben
*******	
C860	BASIC-Stackpointer
C87B	Vergleich HL <> DE
*******	BASIC-Befehl ON
C885	'ERROR'
C88A	8-Bit-Wert holen
C88F	'GOTO'
C894	Test auf nachfolgendes Zeichen
C897	'GOSUB'
C899	nachfolgende Blanks überlesen
C89C	Zähler erniedrigen
C89F	Zeilennummer nach DE holen
C8A2	folgt Komma ?
******	Event-Verarbeitung (AFTER/EVERY)

C8B5	
C8B9	KL NEXT SYNC
C8BC	steht kein Ereignis an ?
C8BE	Priorität merken
C8C2	Bit 7 löschen
C8C8	Adresse des Eventblocks
C8C9	KL DO SYNC
C8D4	KL DONE SYNC
C8D9	nächstes Event
C8E0	Unterbrechung durch 'ESC' erlauben
C8ED	'Break'
C8F2	zur Interpreterschleife
C8FC	ON-BREAK-Adresse
C901	Zeilennummer nach HL
C906	Direktmodus ?
C909	SOUND CONTINUE
C915	Test auf nachfolgendes Zeichen
C918	'GOSUB'
C919	Zeilenadresse holen
C91D	nach BC
C920	10
******	Event-Routine
C929	
C92D	Zeilennummer holen/Direktmodus ?
C932	ja
C934	Kennbyte für AFTER/EVERY-GOSUB
C937	GOSUB-Befehl
C93A	Adresse des aktuellen Statements
C949	Adresse des aktuellen Statements
C95A	-8
C95E	KL DONE SYNC
C968	-4
C96C	KL DONE SYNC
C96F	Unterbrechung durch 'Break' erlauben
C976	zur Interpreterschleife
******	BASIC-Befehl ON BREAK
C979	
C97C	Blanks überlesen
C97F	'CONT'

C984	'STOP'
C986	Defaultwert Null bei Stop
C98B	Test auf nachfolgendes Zeichen
C98E	'GOSUB'
C98F	Zeilenadresse holen
C993	ON-BREAK-Adresse
C997	KM DISARM BREAK
******	BASIC-Befehl DI
C99A	
C99B	KL EVENT DISABLE
******	BASIC-Befehl EI
C9A0	
C9A1	KL EVENT ENABLE
******	SOUND- und Event-Reset
C9A6	SOUND RESET
C9A9	Basisadresse des Eventblocks
C9AC	4 Timer
C9AF	KL DEL TICKER
C9B3	18
C9B6	addieren
C9B7	nächster Timer
C9B9	KM DISARM BREAK
C9BC	KL SYNC RESET
C9C2	ON-BREAK-Adresse löschen
C9C5	Unterbrechung durch BREAK erlauben
C9C8	Adresse der Sound-Queue
C9D4	Adresse des Eventblocks
C9DF	BASIC-ROM-Select
C9E1	ADresse der Event-Routine
C9E4	KL INIT EVENT
******	BASIC-Befehl ON SQ
C9F8	Test auf '('
C9FB	8-Bit-Wert holen
C9FF	Adresse der Sound-Queue berechnen
CA05	Test auf ')'
CA08	'GOSUB' und Adresse holen
CA0E	SOUND ARM EVENT
******	Adresse der Sound-Queue berechnen
CA13	Bit 0 gesetzt ?

CA18	Bit 1 gesetzt ?
CA1D	Bit 2 gesetzt ?
CA22	'Improper Argument'
******	BASIC-Befehl AFTER
CA25	16-Bit-Wert 0 - 32767 holen
CA28	Recharge Count auf Null
******	BASIC-Befehl EVERY
CA2D	16-Bit-Wert 0 - 32767 holen
CA30	als Count und
CA31	Recharge Count
CA34	folgt Komma ?
CA37	Defaultwert Null
CA3A	ja, Integerwert mit Vorzeichen holen
CA3E	aus Timer# Adresse des Eventblocks holen
CA42	6 Bytes für Tickerblock addieren
CA47	'GOSUB' und Adresse holen
CA4E	KL ADD TICKER
******	BASIC-Funktion REMAIN
CA53	CINT
CA56	Adresse des Eventblocks holen
CA59	KL DEL TICKER
CA5C	gefunden ?
CA5E	nein, Null
CA62	Integerzahl in HL übernehmen
******	Adresse des Eventblocks berechnen
CA65	
CA66	Hi-Byte gleich Null ?
CA67	nein, 'Improper argument'
CA6A	größer gleich 4 ?
CA6C	ja, 'Improper argument'
CA70	* 18
CA74	Basisadresse Eventtabelle
CA77	plus Offset
******	zugehöriges NEXT suchen
CA79	
CA7A	aktuelle Zeilenadresse nach HL
CA7F	Zähler für Verschachtelung
CA81	Fehlernummer für 'NEXT missing'
CA87	Blanks überlesen

CA8A	'NEXT'
CA8F	'FOR'
CA93	Verschachtelung erhöhen
CA94	weiter suchen
CA99	aktuelle Zeilenadresse nach HL
CA9D	aktuelle Zeilenadresse setzen
CAA1	Verschachtelung erniedrigen
CAA2	zugehöriges NEXT gefunden ?
CAA4	Blanks überlesen
CAA7	Zeilenende ?
CAAB	Variable suchen
CAB0	folgt Komma ?
CAB3	nein
CAB5	sonst nächste Variable nach NEXT
CAB8	zugehöriges NEXT gefunden
CABA	ja
CABD	aktuelle Zeilenadresse nach HL
CAC1	aktuelle Zeilenadresse setzen
CAC6	weitersuchen
CAC9	Blanks überlesen
******	zugehöriges WEND suchen
CACC	
CACE	aktuelle Zeilenadresse nach HL
CAD2	Zähler für Verschachtelung
CAD4	erhöhen
CAD5	Fehlernummer für 'WEND missing'
CADB	Blanks überlesen
CADF	'WHILE'
CAE1	Verschachtelung erhöhen
CAE3	'WEND'
CAE7	Verschachtelung erniedrigen
CAE9	Blanks überlesen
CAEC	Blanks überlesen
CAEF	Eingabezeile holen
CAF3	Stream 0 selektieren
CAF6	Stackpointer initialisieren
CAF9	zur Interpreterschleife
******	Eingabezeile holen
CAFC	Zeiger auf Eingabepuffer

CAFF	Pufferinhalt löschen
CB01	Eingabezeile holen
******	Zeile editieren
CB04	Zeiger auf Eingabepuffer
CB07	Zeile editieren
CB0A	LF ausgeben
******	Eingabezeile von Diskette holen
CB0D	
CB0E	Zeiger auf Eingabepuffer
CB18	DISK IN CHAR
CB1D	CR
CB25	LF
CB2D	Fehlermeldung ausgeben
CB30	'Line too long'
CB32	LF
******	Fehlernummer löschen
CB3A	
******	Fehlernummer setzen
CB3B	Diskettenfehler
CB3E	Fehlernummer
CB41	aktuelle Zeilenadresse nach HL
CB44	als ERROR-Line
******	Fehlermeldung ausgeben
CB48	Rücksprungadresse nach HL
CB49	Zeichen nach CALL-Befehl holen
CB4A	als Fehlernummer, Meldung ausgeben
******	'Syntax error' ausgeben
CB4C	Fehlernummer für 'Syntax error'
CB4E	Fehlermeldung ausgeben
******	'Improper argument' ausgeben
CB50	Fehlernummer für 'Improper argument'
CB52	Fehlermeldung ausgeben
******	BASIC-Befehl ERROR
CB54	8-Bit-Wert ungleich 0 holen
CB58	Fehlernummer und -zeile setzen
CB5B	Adresse des aktuellen Statements
CB5E	Programmzeiger nach ERROR
CB61	Zeilenadresse und Programmzeiger merken
CB67	Stackpointer auf C000

CB70	Descriptorstack initialisieren
CB79	Adresse der ON-ERROR-Routine
CB7D	Flag für in Fehlerbehandlung
CB8B	zur Interpreterschleife
CB90	Fehlernummer
CB93	Adresse der Fehlermeldung berechnen
CB99	als aktuelle Zeilennummer
CBA3	Diskettenfehler
CBAA	zum READY-Modus
CBAD	Adresse der ERROR-Zeile
CBB0	Zeilennummer nach HL holen
CBBA	Zeiger auf 'Division by zero'
CBBD	Fehlernummer
CBC3	Zeiger auf 'Overflow'
CBC6	Fehlernummer
CBCA	Adresse der ON-ERROR-Routine
CBD0	Fehlernummer in Akku
CBD1	Fehlermeldung ausgeben
CBD4	Streamnummer auf Null
CBD5	Stream selektieren
CBD8	alte Streamnummer merken
CBDA	Fehlermeldung ausgeben
CBDE	LF ausgeben
CBE1	alte Streamnummer
CBE2	selektieren
CBE9	Bildschirm initialisieren
CBEC	'Undefind line' ausgeben
CBEF	Zeilennummer ausgeben
CBF2	'in Zeilennummer' ausgeben
CBF4	'Undefined line ',0
CC04	Zeiger auf 'Break'
CC0A	Bildschirm initialisieren
CC0D	Fehlermeldung ausgeben
CC10	Zeilenadresse holen
CC13	Direktmodus ?
CC15	Zeiger auf ' in '
CC18	String ausgeben
CC1C	Zeilennummer ausgeben
CC1F	'Break'

CC24	' in ',0
******	BASIC-Befehl STOP
CC29	
CC2B	'Break in Zeilennummer' ausgeben
CC32	zum READY-Modus
******	BASIC-Befehl END
CC34	
******	Diskettenfehler
CC3A	Disk-Error merken
CC3D	Fehlermeldung ausgeben
CC40	'Nr 32'
CC4A	zum READY-Modus
CC66	zum READY-Modus
CC6A	Zeilennummer nach HL holen
CC6E	Direktmodus ?
CC70	Ende des Statements ?
CC7B	aktuelle Zeilenadresse setzen
CC87	Direktmodus ?
CC8A	ja
CC8B	Zeilenadresse nach HL
CC8E	Zeilenadresse nach Unterbrechung
CC92	Programmzeiger nach Unterbrechung
******	BASIC-Befehl CONT
CC96	
CC97	Programmzeiger nach Unterbrechung
CC9B	Test auf Direktmodus
CC9C	'Cannot CONTinue'
CC9E	Fehlermeldung ausgeben
CCA2	Zeilenadresse nach Unterbrechung
CCA5	aktuelle Zeilenadresse setzen
CCA8	SOUND CONTINUE
CCAC	zur Interpreterschleife
CCB0	Flag für in Fehlerbehandlung löschen
CCB6	Adresse der ON-ERROR-Routine
******	ON ERROR
CCBB	Blanks überlesen
CCBE	Test auf nachfolgendes Zeichen
CCC1	'GOTO'
CCC2	Zeilennummer nach DE holen

CF0F	Typ String, sonst 'Type mismatch'
******	Zeilennummernbereich holen
CF12	1 und
CF15	65535 als Default
CF18	folgt Komma ?
CF1B	nein, Ende des Statements ?
CF1E	ja
CF1F	'#'
CF22	'-'
CF26	Zeilennummer nach DE holen
CF2A	und nach BC
CF2C	folgt Komma ?
CF2F	ja
CF30	Test auf nachfolgendes Zeichen
CF33	'-'
CF34	65535 als Default-Endwert
CF38	folgt Komma ?
CF3B	ja
CF3C	Zeilennummer nach DE holen
CF3F	folgt Komma ?
CF46	'Improper argument'
******	Zeilennummer nach DE holen
CF4B	Konstantentyp
CF4E	Wert nach DE
CF50	Zeilennummer ?
CF52	ja, fertig
CF54	Zeilenadresse ?
CF56	nein, 'Syntax error'
CF5A	HL zeigt auf Zeilenbeginn
CF5F	Zeilennummer nach DE
CF62	Blanks überlesen
******	Ausdruck holen
CF65	
CF66	Hierarchie-Kode Null
CF68	Term holen
CF6D	Blanks überlesen
******	Term holen
CF70	
CF72	Ausdruck holen

CF78	Operator
CF79	’>’
CF7B	kleiner ?
CF7C	’NOT’
CF7E	größer gleich ?
CF7F	’+’
CF81	kleiner, dann Vergleichsoperator
CF83	’+’, dann Test auf String
CF86	kein String
CF8A	Stringdescriptor
CF8D	auf Stack
CF8E	Ausdruck holen
CF91	Typ String, sonst ’Type mismatch’
CF95	Stringaddition
CF98	nächsten Term bearbeiten
******	arithmetische Operatoren
CF9A	
CF9B	minus #F4
CF9E	mal 4
CFA2	plus #CFF0, Tabellenadresse
CFA9	Hierarchie-Kode
CFAB	kleiner, fertig
CFAD	Ergebnis auf Stack ablegen
CFB3	Hierarchie-Kode
CFB4	Term holen
CFC0	Platz im BASIC-Stack reservieren
CFC3	JP (DE), Operation durchführen
CFC6	nächsten Term bearbeiten
******	Vergleichsoperatoren
CFC8	
CFCD	Token
CFCE	minus Offset
CFD1	Test auf String
CFD4	Adresse für arithmetische Vergleiche
CFD7	kein String
CFDA	Stringdescriptor
CFDD	auf Stack
CFDF	Hierarchie-Kode
CFE1	Term holen

CFE7	Stringvergleich
CFEB	Ergebnis des Vergleichs holen
CFEE	nächsten Term bearbeiten
******	BASIC-Operatoren Hierarchie-Kodes+Adressen
CFF0	F4, '+'
CFF3	F5, '-'
CFF6	F6, '*'
CFF9	F7, '/'
CFFC	F8, '^'
CFFF	F9, 'Backslash'
D002	FA, 'AND'
D005	FB, 'MOD'
D008	FC, 'OR'
D00B	FD, 'XOR'
******	arithmetischer Vergleich
D00E	
D012	arithmetischer Vergleich
D01D	Vorzeichen übernehmen
******	'-' negatives Vorzeichen
D020	Hierarchie-Kode
D022	Term holen
D026	Vorzeichen wechseln
******	BASIC-Operator NOT
D02B	Hierarchie-Kode
D02D	Term holen
D031	NOT-Operator
******	Ausdruck holen
D036	Blanks überlesen
******	Ausdruck holen
D039	'Operand missing'
D03D	Variable holen
D041	numerischen Wert holen
D043	'"'
D045	String holen
D048	Funktion ?
D04A	zur Funktionsberechnung
D04E	Basisadresse der Tabelle
D051	Tabelle durchsuchen
D055	Blanks überlesen, Sprung auf Funktion

D058	Fehlermeldung ausgeben
D05B	'Operand missing'
******	Sonderfunktionen
D05C	Anzahl der Tabelleneinträge
D05D	nicht gefunden, 'Syntax error'
D05E	'-'
D062	'+'
D065	'('
D068	'NOT'
D06B	'ERL'
D06E	'FN'
D071	'MID$'
D074	'ß'
******	Variable holen
D077	Variablenadresse holen
D07A	noch nicht angelegt ?
D07C	Variablentyp
D07E	String ?
D087	String ?
D089	Variable löschen
D08C	Zeiger auf Null
D08F	als Stringdescriptor
D094	Stringlänge Null
******	numerischen Wert holen
D095	Offset abziehen
D09A	kleiner 10 ?
D09C	ja, Ziffer holen
D0A0	Ein-Byte-Wert ?
D0A2	ja
D0A6	Zwei-Byte-Wert (dez, hex, bin) ?
D0A8	ja
D0AA	Fließkommawert ?
D0AC	ja
D0AE	'Syntax error'
D0B1	'Real'
D0B3	Variablentyp setzen
******	Zwei-Byte-Wert holen
D0B9	
******	Fließkommawert holen

D0C0	
D0CD	Variablentyp auf 'Real'
D0D1	Blanks überlesen
******	'(' Term in Klammern holen
D0D4	Ausdruck holen
D0D7	Test auf ')'
******	
D0DA	'Syntax error'
******	Funktionsberechnung
D0DD	Programmzeiger erhöhen
D0DE	Token holen
D0DF	Blanks überlesen
D0E2	Token testen
D0E5	40 - 49, reservierte Variable
D0E9	Adresse der reservierten Variablen holen
D0EC	Test auf '('
D0F0	Token mal 2
D0F6	'Syntax error'
D0FA	Funktion berechnen
D0FC	Funktionsargument in Klammern holen
D100	Funktion berechnen
******	Funktion berechnen
D105	Adresse der Funktionen
D10A	Hi-Byte löschen
D10C	Token mal 2 addieren
D111	Funktion ausführen
******	Adresse der reservierten Variablen holen
D113	Token verdoppeln
D115	Basisadresse der Tabelle-Offset
******	Adressen der reservierten Variablen
D11A	40, EOF
D11C	41, ERR
D11E	42, HIMEM
D120	43, INKEY$
D122	44, PI
D124	45, RND
D126	46, TIME
D128	47, XPOS
D12A	48, YPOS

D12C	49, DERR
******	reservierte Variable DERR
D12E	Disk-Error-Nummer
******	reservierte Variable ERR
D133	ERROR-Nummer
D137	Akkuinhalt als Integerzahl übernehmen
******	reservierte Variable TIME
D13D	KL TIME PLEASE
D140	4-Byte-Wert nach Fließkomma wandeln
******	reservierte Variable ERL
D146	ERROR-Zeilennummer holen
******	reservierte Variable HIMEM
D14B	
D14C	HIMEM
D14F	Wert übernehmen
******	'ß', Variablenpointer
D151	Variablenadresse holen
D154	nicht definiert, 'Improper argument'
D15A	String ?
D15F	Wert übernehmen
******	reservierte Variable XPOS
D164	
D165	GRA ASK CURSOR
D168	Spaltenwert nach HL
******	reservierte Variable YPOS
D16C	GRA ASK CURSOR
D16F	Integerzahl in HL übernehmen
*******	BASIC-Befehl DEF
D174	Test auf nachfolgendes Zeichen
D177	'FN'
D179	Zeilennummer nach HL holen
D17D	'Invalid direct command'
D17F	Fehlermeldung ausgeben
D182	Funktion suchen
D18A	Rest des Statements überlesen
******	BASIC-Funktion FN
D18D	Funktion suchen
D199	'Unknown user function'
D19B	Fehlermeldung ausgeben

D1A2	'('
D1A6	Blanks überlesen
D1AA	Test auf '('
D1B3	Ausdruck holen
D1B8	Wert an Variable zuweisen
D1BC	folgt Komma ?
D1BF	nein
D1C2	Test auf ','
D1C5	nächste Variable
D1C7	Test auf ')'
D1CB	Test auf ')'
D1D1	Test auf '='
D1D4	Ausdruck holen
D1D7	'Syntax error'
D1DA	Test auf String
D1E5	auf gleichen Variablentyp prüfen
******	BASIC-Funktionen mit mehreren Argumenten
D1E8	71, BIN$
D1EA	72, DEC$
D1EC	73, HEX$
D1EE	74, INSTR
D1F0	75, LEFT$
D1F2	76, MAX
D1F4	77, MIN
D1F6	78, POS
D1F8	79, RIGHT$
D1FA	7A, ROUND
D1FC	7B, STRING$
D1FE	7C, TEST
D200	7D, TESTR
D202	7E, COPYCHR$
D204	7F, VPOS
******	Adressen der BASIC-Funktionen
D206	00, ABS
D208	01, ASC
D20A	02, ATN
D20C	03, CHR$
D20E	04, CINT
D210	05, COS

D212	06, CREAL
D214	07, EXP
D216	08, FIX
D218	09, FRE
D21A	0A, INKEY
D21C	0B, INP
D21E	0C, INT
D220	0D, JOY
D222	0E, LEN
D224	0F, LOG
D226	10, LOG10
D228	11, LOWER$
D22A	12, PEEK
D22C	13, REMAIN
D22E	14, SGN
D230	15, SIN
D232	16, SPACE$
D234	17, SQ
D236	18, SQR
D238	19, STR$
D23A	1A, TAN
D23C	1B, UNT
D23E	1C, UPPER$
D240	1D, VAL
******	BASIC-Funktion MIN
D242	Flag für MIN
******	BASIC-Funktion MAX
D246	Flag für MAX
D248	Ausdruck holen
D24B	folgt Komma ?
D24E	nein, Test auf ')', fertig
D251	Variable auf BASIC-Stack ablegen
D254	Ausdruck holen
D259	Platz im BASIC-Stack freigeben
D25E	arithmetischer Vergleich
D267	Ergebnis des Vergleichs holen
D26B	nächstes Argument
******	BASIC-Funktion ROUND
D26D	Ausdruck holen

D270	und auf BASIC-Stack ablegen
D273	folgt Komma ?
D276	Default Null
D279	ja, Integerwert mit Vorzeichen holen
D27C	Test auf ')'
D281	39
D284	addieren
D285	79
D288	Vergleich HL <> DE
D28B	größer, 'Improper argument'
D290	Platz im BASIC-Stack freigeben
D293	Rundungsstellenzahl nach B
D294	Zahl runden
******	BASIC-Befehl CAT
D29B	Disk I/O abbrechen
D2A1	DISK CATALOG
D2A4	Diskettenfehler merken
******	BASIC-Befehl OPENOUT
D2AB	
D2B4	DISK OUT OPEN
******	BASIC-Befehl OPENIN
D2B7	
D2BD	Fehlermeldung ausgeben
D2C0	'File type error'
D2C1	Filenamen holen
D2C7	DISK IN OPEN
D2CD	Stringausdruck und -parameter holen
D2D2	Test auf Systemmeldungen
D2D5	Diskettenfehler merken
D2DA	Fehlermeldung ausgeben
D2DD	'File already open'
******	Test auf Systemmeldungen
D2DE	
D2E0	kein Filenamen ?
D2E3	erstes Zeichen des Namens
D2E4	'!'
D2E8	nein
D2EA	Zeiger auf zweites Zeichen setzen
D2EB	Länge erniedrigen

D2EC	Flag umkehren
D2ED	CAS NOISY
******	BASIC-Befehl CLOSEIN
D2F0	
D2F1	DISK IN CLOSE
******	BASIC-Befehl CLOSEOUT
D2F8	
D2F9	DISK OUT CLOSE
D2FC	Diskettenfehler merken
******	Disk I/O abbrechen
D303	
D306	DISK IN ABANDOM
D30C	DISK OUT ABANDOM
*******	BASIC-Befehl SOUND
D316	8-Bit-Wert holen
D319	Kanal-Status
D31C	Test auf ','
D31F	Argument 0 bis 4095 holen
D322	Ton-Periode
D326	folgt Komma ?
D329	Defaultwert 20
D32C	ja, Integerwert mit Vorzeichen holen
D32F	Dauer
D333	max. 15, Default 12
D336	falls vorhanden Argumente holen
D339	Lautstärke
D33C	max. 15, Default 0
D33E	falls vorhanden Argumente holen
D341	Lautstärken-Hüllkurve
D344	falls vorhanden Argumente holen
D347	Ton-Hüllkurve
D34A	max. 31, Default 0
D34C	falls vorhanden Argumente holen
D34F	Geräusch-Periode
D352	Ende des Statements, sonst 'Syntax error'
D356	Adresse des Sound-Parameterblocks
D359	SOUND QUEUE
D35F	zur Interpreterschleife
******	falls vorhanden 8-Bit-Wert holen

D362	folgt Komma ?
D365	Defaultwert laden
D366	kein Komma, fertig
D368	','
D36C	8-Bit-Wert holen
D36F	mit Maximalwert vergleichen
D370	kleiner, ok
D371	'Improper argument'
******	BASIC-Befehl RELEASE
D373	8
D375	8-Bit-Wert < 8 holen
D379	SOUND RELEASE
******	BASIC-Funktion SQ
D37E	CINT
D383	Bit 0 testen
D384	gesetzt ?
D386	Bit 1 testen
D387	gesetzt ?
D389	Bit 2 testen
D38A	nicht gesetzt, 'Improper argument'
D38C	Hi-Byte größer Null ?
D38D	ja, 'Improper argument'
D390	SOUND CHECK
D393	Akkuinhalt als Integerzahl übernehmen
******	Argument -128 bis +127 holen
D396	Integerwert mit Vorzeichen holen
D39E	'Improper argument'
******	BASIC-Befehl ENV
D3A1	8-Bit-Wert ungleich Null holen
D3A4	größer gleich 16 ?
D3A6	ja, 'Improper argument'
D3AC	Parameter holen
D3B1	Adresse des Parameterblocks
D3B5	SOUND AMPL ENVELOPE
D3BB	'='
D3BF	Blanks überlesen
D3C2	16
D3C4	8-Bit-Wert < 16 holen
D3C7	Bit 7 setzen

D3CA	Test auf ','
D3CD	16-Bit-Wert holen
D3D0	128
D3D2	8-Bit-Wert < 128 holen
D3D5	2 Argumente holen
******	BASIC-Befehl ENT
D3D7	Argument -128 bis +127 holen
D3DD	Null ?
D3E2	Null ?
D3E3	'Improper argument'
D3E5	größer gleich 16 ?
D3E7	'Improper argument'
D3ED	Parameter holen
D3F2	Adresse des Parameterblocks
D3FB	SOUND TONE ENVELOPE
D401	'='
D405	Blanks überlesen
D408	Argument 0 bis 4095 holen
D412	240
D414	8-Bit-Wert < 240 holen
D418	Test auf ','
D41B	Argument -128 bis +127 holen
D41F	Test auf ','
D422	8-Bit-Wert holen
******	Parameter für ENT & ENV holen
D428	
D42B	folgt Komma ?
D432	JP (DE)
D439	Adresse des Parameterblocks
D44C	Ende des Statements, sonst 'Syntax error'
******	Argument 0 bis 4095 holen
D44F	Integerwert mit Vorzeichen holen
D452	Hi-Byte
D453	Bit 12-15 gesetzt ?
D455	ja, 'Improper argument'
******	BASIC-Funktion INKEY
D459	CINT
D45C	80
D45F	Vergleich HL <> DE

D462	'Improper argument'
D465	KM TEST KEY
D468	-1 wenn nicht gedrückt
D46D	Ergebnis nach L
D470	Integerzahl in HL übernehmen
******	BASIC-Funktion JOY
D473	KM GET JOYSTICK
D477	CINT
D483	Akkuinhalt als Integerzahl übernehmen
D486	'Improper argument'
******	BASIC-Befehl KEY
D489	'DEF'
D48D	8-Bit-Wert holen
D491	Test auf ','
D494	Stringausdruck und Parameter holen
D497	Stringlänge nach C
D499	Tastennummer nach B
D49B	Stringadresse nach HL
D49C	KM SET EXPAND
D4A0	'Improper argument'
******	KEY DEF
D4A3	Blanks überlesen
D4A6	80 als Maximalwert
D4A8	8-Bit-Wert < 80 holen
D4AC	Test auf ','
D4AF	2
D4B1	Argument < 2 holen
D4BA	KM SET REPEAT
D4BF	KM SET TRANSLATE
D4C2	auf weiteres Argument testen
D4C5	KM SET SHIFT
D4C8	auf weiteres Argument testen
D4CB	KM SET CONTROL
D4CE	folgt Komma ?
D4D1	nein, fertig
D4D3	8-Bit-Wert holen
D4D9	Sprung nach (HL)
******	BASIC-Befehl SPEED
D4DE	'WRITE'

D4E2	'KEY'
D4E4	KM SET DELAY
D4E9	'INK'
D4EB	SCR SET FLASHING
D4EE	'Syntax error'
******	SPEED KEY & INK
D4F1	
D4F2	Blanks überlesen
D4F5	8-Bit-Wert ungleich Null holen
D4F9	Test auf ','
D4FC	8-Bit-Wert ungleich Null holen
D503	Sprung nach (BC)
******	SPEED WRITE
D508	Blanks überlesen
D50B	2
D50D	Argument < 2 holen
D511	167
D517	Null ?
D519	nein, Zeitkonstante verdoppeln
D51B	CAS SET SPEED
******	Reservierte Variable PI
D520	
D521	Typ auf 'Real' setzen
D524	Variablentyp nach C, HL auf Variable
D527	PI holen
******	BASIC-Befehl DEG
D52C	Flag für DEG
******	BASIC-Befehl RAD
D530	Flag für RAD
D531	DEG/RAD-Modus setzen
******	BASIC-Funktion SQR
D534	SQR-Funktion
******	BASIC-Operator '^'
D539	
D53B	CREAL
D53F	Zwischenspeicher für Fließkommavariable
D542	Variable von (DE) nach (HL) kopieren
D548	Variablentyp und Adresse setzen
D54C	Potenzierung

D54F	Funktion ausführen
D552	fehlerfrei ?
D553	'Division by zero'
D556	'Overflow'
D559	'Improper argument'
******	Fließkommafunktion ausführen
D55C	
D55E	CREAL
D562	Funktion ausführen
******	BASIC-Funktion EXP
D563	EXP-Funktion
******	BASIC-Funktion LOG10
D568	LOG10-Funktion
******	BASIC-Funktion LOG
D56D	LOG-Funktion
******	BASIC-Funktion SIN
D572	SIN-Funktion
******	BASIC-Funktion COS
D577	COS-Funktion
******	BASIC-Funktion TAN
D57C	TAN-Funktion
******	BASIC-Funktion ATN
D581	ATN-Funktion
D586	'Random number seed ? ',0
******	BASIC-Befehl RANDOMIZE
D59C	
D59E	Ausdruck holen
D5A5	'Random number seed ? '
D5A8	ausgeben
D5AB	Eingabezeile holen
D5AE	LF ausgeben
D5B1	Eingabe lesen
D5B4	ungültig, wiederholen
D5B6	Blank, TAB und LF überlesen
D5BA	ungültig, wiederholen
D5BC	CREAL
D5BF	SET RANDOM SEED
******	reservierte Variable RND
D5C4	

D5C5	'('
D5C9	Blanks überlesen
D5CC	Ausdruck holen
D5CF	Test auf ')'
D5D3	CREAL
D5D6	SGN
D5D9	ungleich Null ?
D5DB	letzten RND-Wert holen
D5E0	negativ, SET RANDOM SEED
D5E5	Typ auf Fließkomma setzen
D5E8	RND
******	Variablenzeiger rücksetzen
D5ED	Tablelle löschen
D5F0	Programmende
D5F3	Variablenstart
D5F6	Arraystart
D5F9	Arrayende
******	Tabelle löschen
D5FD	Basis der Tabelle
D600	54 = 2*27, A-Z plus Funktionen
D602	#ADB7 bis #ADEC löschen
D60A	#ADED bis #ADF2 löschen
******	Flag für FN löschen
D611	
******	Tabellenadresse berechnen
D61A	'Z'+1
D61C	Variablenstart
D620	minus 1
D621	mal 2
D624	plus #AD35
******	Tabellenadresse für Array berechnen
D62A	Arraystart
D62E	minus 1
D632	mal
D635	plus #ADED
******	Alle Variablen auf Typ REAL
D63B	'AZ'
D63E	Typ 'Real'
D641	Anzahl nach A

D642	kleiner 1, dann 'Syntax error'
D646	Basis der Tabelle gleich #ADB2
D64B	Buchstabe gleich Zeiger in Tabelle
D64E	alle Buchstaben
******	BASIC-Befehl DEFSTR
D653	Typ 'String'
******	BASIC-Befehl DEFINT
D657	Typ 'Integer'
******	BASIC-Befehl DEFREAL
D65B	Typ 'Real'
D65D	Buchstabe holen
D65E	Test auf Buchstabe
D661	'Syntax error'
D663	nach BC (von - bis)
D665	Blanks überlesen
D668	'-'
D66C	Blanks überlesen
D66F	Test auf Buchstabe
D672	'Syntax error'
D674	bis
D675	Blanks überlesen
D678	Variablentyp setzen
D67B	folgt Komma ?
D67E	ja, weitermachen
D681	'Syntax error'
D684	Fehlermeldung ausgeben
D687	'Subscript out of range'
D688	Fehlermeldung ausgeben
D68B	'Array already dimensioned'
D68C	2* #7C 'l'
D68E	Befehlserweiterung
******	BASIC-Befehl LET
D691	Variable holen
D695	Test auf '='
D698	Ausdruck holen
D69D	Wert an Variable zuweisen
******	Wert an Variable zuweisen
D6A2	Variablentyp
D6A3	und Ergebnistyp

D6A6	vergleichen
D6A8	Typanpassung, sonst 'Type mismatch'
D6AB	Test auf String
D6AE	nein, Variable nach (HL) kopieren
D6B2	Stringverwaltung
D6B6	Zeiger auf String übernehmen
******	BASIC-Befehl DIM
D6B9	Dimensionierung
D6BC	folgt Komma ?
D6BF	ja, nächste Variable
******	Variable suchen
D6C2	Variablenname lesen
D6C5	Test auf dimensionierte Variable
D6C8	Variablentyp holen
******	Variablenadresse holen
D6CC	Variablenname lesen
D6CF	Test auf dimensionierte Variable
D6D2	Variablentyp holen
D6D5	erster Buchstabe
D6D6	Tabellenposition berechnen
******	Funktion suchen
D6DE	Variablenname lesen
D6E4	Tabellenposition für FN berechnen
D6EA	Funktion anlegen
D6EF	Variablenname lesen
D6F5	erster Buchstabe
D6F6	Tabellenposition berechnen
D6FC	Variablentyp
D705	Variablenstart
D70C	Variablentyp
D712	Variablenname lesen
D715	Test auf indizierte Variable
D718	Variablentyp holen
D72A	Array suchen
D72E	gefunden ?
D733	Array suchen
D736	gefunden ?
******	Array suchen
D75B	Variablentyp

D77A	Bit 6 setzen, 'FN'
D787	Arraystart
D78B	Platz im Variablenbereich reservieren
D78E	Zeiger für Arraybereich erhöhen
D7C6	LDIR
D7C9	Variablentyp
******	Dimensionierung
D7E4	Variablenname holen
D7E8	'('
D7EC	'['
D7EE	'Syntax error'
D7F6	Variablentyp
D7F9	Tabellenposition für Array berechnen
D7FC	Array suchen
D7FF	gefunden, 'Array already dimensioned'
******	Test auf dimensionierte Variable
D80A	
D80C	'('
D810	'['
D817	Variablenstart
******	dimensionierte Variable
D820	
D827	Arraystart
D830	Variablentyp
D833	Tabellenposition für Array berechnen
D836	Array suchen
D839	nicht gefunden ?
D845	'Subscript out of range'
D857	Anzahl der Dimensionen
D85C	Arraygrenze nach DE
******	Indices lesen
D887	
D888	Blanks überlesen
D88B	Variablentyp
D88E	merken
D88F	Anzahl der Indices
D891	16-Bit-Wert 0 - 32767 holen, Index
D897	Platz im BASIC-Stack reservieren
D89B	Index auf BASIC-Stack

D89E	Anzahl der Indices erhöhen
D89F	folgt Komma ?
D8A2	ja, nächster Index
D8A5	')'
D8A9	']'
D8AB	'Syntax error'
D8AE	Blanks überlesen
D8B2	Variablentyp wiederherstellen
D8C4	Arrayende
D8C8	Platz im Variablenbereich reservieren
D8D5	Variablentyp
D8E3	10, Defaultwert für Index
D92B	2 Bytes
D92D	Platz im BASIC-Stack freigeben
******	Variablenname lesen
D935	Variablentyp feststellen
D93D	Variable schon angelegt ?
D93E	nein
D941	Buchstaben des Namens überlesen
D942	Bit 7 testen
D943	letzter Buchstabe ?
D945	nachfolgende Blanks überlesen
D94B	Zeiger auf Variablentyp setzen
D988	Variablentyp
D991	#05 + #09 => #0D
D995	40
D997	Platz im BASIC-Stack reservieren
D99B	41
D99D	schon 40 Zeichen ?
D99E	ja, dann 'Syntax error'
D9A1	nächstes Zeichen aus Namen lesen
D9A3	Klein- in Großschreibung wandeln
D9A7	letztes Zeichen ?
D9A8	nein
D9AA	BASIC-Stackpointer setzen
D9B0	nachfolgende Blanks überlesen
******	Variablentyp feststellen
D9B3	
D9B6	kleiner #0B ?

D9B8	-#09, #0D => #05
D9BA	'!', Real-Variable ?
D9BC	Typ auf 'Real' setzen
D9BE	'Syntax error'
D9C0	'%', Integer-Variable ?
D9C2	oder '$', String ?
D9C4	nein, 'Syntax error'
D9C7	'Real'
D9C9	Variablentyp merken
******	Arraytabelle updaten
D9CD	Tabelle für Arrays löschen
D9D0	Arrayende
D9D4	Arraystart
D9D7	Vergleich HL <> DE
D9DA	keine Arrays
D9E5	Tabellenposition für Array berechnen
******	BASIC-Befehl ERASE
D9F4	
D9F7	Array löschen
D9FA	folgt Komma ?
D9FD	ja, nächstes Array
******	Array löschen
DA00	Variablenname lesen
DA04	Variablentyp
DA07	Tabellenposition für Array berechnen
DA0A	Array suchen
DA0D	nicht gefunden, 'Improper argument'
DA16	BC := HL - DE
DA1F	Arraytabelle updaten
DA33	6 Bytes
DA35	Platz im BASIC-Stack reservieren
DA71	Platz im BASIC-Stack reservieren
DA75	Variablentyp feststellen
DA88	Variablentyp
DA8D	Platz im BASIC-Stack reservieren
DAAC	26 Buchstaben, 'A'
DAB0	erster Buchstabe des Namens
DAB1	Tabellenposition berechnen
DAB8	nächster Buchstabe

DAB9	schon alle Buchstaben ?
DABD	Tabellenposition für Array berechnen
DAC8	Arraystart
DAE2	Vergleich HL <> BC
DB10	Sprung nach (HL)
******	BASIC-Befehl LINE
DB18	Test auf nachfolgendes Zeichen
DB1B	'INPUT'
DB1C	Kanalnummer holen
DB1F	evtl. Dialogstring ausgeben
DB22	Variable suchen
DB25	Type 'String', sonst 'Type mismatch'
DB2A	Eingabe vom aktiven Gerät holen
DB2D	String in Descriptorstack eintragen
DB31	Ergebnis an Vaeriable zuweisen
******	Eingabe von aktivem Gerät holen
DB36	
DB39	Eingabe von Diskette holen
DB42	';'
******	BASIC-Befehl INPUT
DB48	Kanalnummer holen
DB4B	Eingabe holen und umwandeln
DB4F	Variable suchen
DB59	folgt Komma ?
DB5C	ja, nächste Variable
******	Eingabe holen und umwandeln
DB60	
DB63	evtl. Dialogstring ausgeben
******	
DB7E	'?Redo from start', LF,0
******	evtl. Dialogstring ausgeben
DB90	
DB91	';'
DB93	Trennzeichen merken
DB96	Blanks überlesen
DB99	'"'
DB9B	kein String ?
DBA0	folgt Komma ?
DBA3	ja

DBA4	Test auf nachfolgendes Zeichen
DBA7	';'
DBAC	'?'
DBAE	ausgeben
DBB1	' '
DBB3	ausgeben
DBC6	'Type mismatch'
DBCE	Fehlermeldung ausgeben
DBD1	'Type mismatch'
DBD5	Variablenname und -typ holen
DBDF	'String'
DBE1	ja, Stringparameter holen
DBE5	folgt Komma ?
DBEA	nein
DBEC	','
DBFD	Test auf String
DC13	Blank, TAB und LF überlesen
DC22	String in Descriptor eintragen
DC25	Blank, TAB und LF überlesen
DC28	'"'
DC2A	String lesen
DC39	Fehlermeldung ausgeben
DC3C	'EOF met'
DC42	'"'
DC53	Start des Eingabepuffers
DC56	erstes Zeichen gleich Null
DC59	'"'
DC5F	'EOF met'
DC64	Start des Eingabepuffers
DC6A	JP (DE)
DC7F	CR
DC82	'"'
DCA4	LF + CR
DCA7	CR ?
DCAA	LF ?
DCBE	','
DCC1	CR
DCC4	' '
DCC7	TAB

DCCA	LF
******	BASIC-Befehl RESTORE
DCCD	keine Zeilennummer ?
DCCF	Zeilennummer nach DE holen
DCD3	BASIC-Zeile DE suchen
DCD7	DATA-Zeiger setzen
DCDA	Programmstart
DCDD	als DATA-Zeiger
******	BASIC-Befehl READ
DCDF	
DCE0	DATA-Zeiger
DCE3	nächstes DATA-Element holen
DCE7	Variable suchen
DCEC	':'
DCF6	','
DCFA	Zeilenadresse während READ-Befehl
DCFD	aktuelle Zeilenadresse setzen
DD00	'Syntax error'
DD04	folgt Komma ?
DD08	ja
DD0A	DATA-Zeiger
DD10	','
DD13	Rest der Zeile überlesen
DD16	Zeilenende ?
DD17	nein
DD1A	Zeilenlänge
DD1C	Null, Programmende ?
DD1E	'DATA exhausted'
DD20	Fehlermeldung ausgeben
DD23	Zeilenadresse während READ-Befehl
DD27	Blanks überlesen
DD2A	'DATA'
DD2C	nein, weitersuchen
DD2F	Vorzeichen merken
DD30	Absolutwert bilden
******	Vorzeichen B übernehmen
DD3C	
DD41	Vorzeichen des Ergebnisses
DD42	negativ, dann Vorzeichenwechsel

DD47	Vorzeichenbit umkehren
******	Integer-Addition HL := DE + HL
DD4F	Carry-Flag löschen
DD50	Addition
DD53	Ergebnis positiv ?
DD54	Flags setzen
******	Integer-Subtraktion HL := DE - HL
DD57	Operanden vertauschen
DD58	Carry-Flag löschen
DD59	Subtraktion
DD5C	Ergebnis positiv ?
DD5D	Flags setzen
******	Integer-Multiplikation mit Vorzeichen
DD60	Vorzeichen des Ergebnisses bestimmen
DD63	vorzeichenlose Multiplikation
DD66	Vorzeichen übernehmen
******	Vorzeichen des Ergebnisses bestimmen
DD6C	Vorzeichen von HL
DD6D	und Vorzeichen von DE
DD6E	nach B
DD70	Absolutwert von DE bilden
DD74	Absolutwert von HL bilden
******	Integermultiplikation ohne Vorzeichen
DD77	
******	Integer-Division mit Vorzeichen
DDA1	Division HL := HL / DE
DDA4	Vorzeichen übernehmen
******	Integer-MOD-Berechnung
DDA8	Vorzeichen merken
DDA9	Division
DDAC	Rest nach HL
DDAD	Vorzeichen zurück holen
DDAE	und übernehmen
******	Division HL := HL / DE, DE := Rest
DDB0	Vorzeichen des Ergebnisses bestimmen
******	Absolutwert bilden
DDEF	Vorzeichen testen
DDF1	positiv, schon fertig
******	Integer-Vorzeichenwechsel

DDF2	
******	SGN Vorzeichen von HL
DDFE	
******	Vergleich HL <> DE
DE07	Vorzeichen von HL
DE08	und Vorzeichen von DE
DE0A	Zahlen mit gleichem Vorzeichen vergleichen
******	Test auf nachfolgendes Komma
DE1A	','
******	Test auf Klammer auf
DE1E	'('
******	Test auf Klammer zu
DE22	')'
******	Test auf Gleichheitszeichen
DE26	'='
******	Test auf nachfolgendes Zeichen
DE2A	Rücksprungadresse holen
DE2B	Zeichen holen
DE2D	Programmzeiger zurückholen
DE2E	Zeichen vergleichen
DE2F	'Syntax error'
******	Blanks überlesen
DE31	
DE33	' '
DE35	weiter auf Blanks testen
DE37	Ende des Statements ?
******	Test auf Zeilenende, sonst 'Syntax error'
DE3C	
DE40	'Syntax error'
******	Test auf Ende des Statements
DE42	
******	Test auf Komma
DE46	
DE47	Blanks überlesen
DE4A	','
DE4D	Blanks überlesen
******	Blank, TAB und LF überlesen
DE52	Zeichen holen
DE53	Zeiger erhöhen

DE54	' '
DE58	TAB
DE5C	LF
DE60	Zeiger erniedrigen
******	Interpreterschleife
DE62	Adresse des aktuellen Statements
DE65	Adresse des aktuellen Statements setzen
DE68	KL POLL SYNCHRONOUS
DE6B	Event-Verarbeitung (AFTER/EVERY)
DE6E	Blanks überlesen
DE71	BASIC-Befehl ausführen
DE74	Programmtext lesen
DE75	':', Ende des Statements ?
DE77	ja
DE79	'Syntax error'
DE7D	Zeilenlänge
DE7E	gleich Null ?
DE80	ja, zum END-Befehl
DE82	aktuelle Zeilenadresse merken
DE86	TRACE-Flag gesetzt ?
DE8A	nein
DE8C	TRACE-Routine
DE8F	zum Anfang der Interpreterschleife
DE91	zum END-Befehl
******	BASIC-Befehl ausführen
DE94	Token mal 2
DE95	Test auf Befehlserweiterung
DE9A	ungültiges Token, 'Syntax error'
DE9F	plus #DEE5 (Tabellenadresse)
DEA7	Befehlsadresse auf Stack
DEA9	Blanks überlesen, Sprung auf Befehl
DEAC	'Syntax error'
******	aktuelle Zeilenadresse auf Null
DEAF	Null
DEB2	als aktuelle Zeilenadresse
******	aktuelle Zeilenadresse laden
DEB6	aktuelle Zeilenadresse
******	Test Direktmodus/Zeilenadresse holen
DEBA	aktuelle Zeilenadresse

DEBF	Null, Direktmodus
DEC2	Zeilennummer nach HL
******	BASIC-Befehl TRON
DEC6	Flag setzen
******	BASIC-Befehl TROFF
DECA	Flag löschen
******	TRACE-Routine
DECF	'['
DED1	ausgeben
DED5	aktuelle Zeilenadresse
DED9	Zeilennummer nach HL
DEDC	Zeilennummer ausgeben
DEE0	']'
DEE2	ausgeben
******	Adressen der BASIC-Befehle
DEE5	80, AFTER
DEE7	81, AUTO
DEE9	82, BORDER
DEEB	83, CALL
DEED	84, CAT
DEEF	85, CHAIN
DEF1	86, CLEAR
DEF3	87, CLG
DEF5	88, CLOSEIN
DEF7	89, CLOSEOUT
DEF9	8A, CLS
DEFB	8B, CONT
DEFD	8C, DATA
DEFF	8D, DEF
DF01	8E, DEFINT
DF03	8F, DEFREAL
DF04	90, DEFSTR
DF07	91, DEG
DF09	92, DELETE
DF0B	93, DIM
DF0D	94, DRAW
DF0F	95, DRAWR
DF11	96, EDIT
DF13	97, ELSE

DF15	98, END
DF17	99, ENT
DF19	9A, ENV
DF1B	9B, ERASE
DF1D	9C, ERROR
DF1F	9D, EVERY
DF21	9E, FOR
DF23	9F, GOSUB
DF25	A0, GOTO
DF27	A1, IF
DF29	A2, INK
DF2B	A3, INPUT
DF2D	A4, KEY
DF2F	A5, LET
DF31	A6, LINE
DF33	A7, LIST
DF35	A8, LOAD
DF37	A9, LOCATE
DF39	AA, MEMORY
DF3B	AB, MERGE
DF3D	AC, MID$
DF3F	AD, MODE
DF41	AE, MOVE
DF43	AF, MOVER
DF45	B0, NEXT
DF47	B1, NEW
DF49	B2, ON
DF4B	B3, ON BREAK
DF4D	B4, ON ERROR GOTO 0
DF4F	B5, ON SQ
DF51	B6, OPENIN
DF53	B7, OPENOUT
DF55	B8, ORIGIN
DF57	B9, OUT
DF59	BA, PAPER
DF5B	BB, PEN
DF5D	BC, PLOT
DF5F	BD, PLOTR
DF61	BE, POKE

DF63	BF, PRINT
DF65	C0, '
DF67	C1, RAD
DF69	C2, RANDOMIZE
DF6B	C3, READ
DF6D	C4, RELEASE
DF6F	C5, REM
DF71	C6, RENUM
DF73	C7, RESTORE
DF75	C8, RESUME
DF77	C9, RETURN
DF79	CA, RUN
DF7B	CB, SAVE
DF7D	CC, SOUND
DF7F	CD, SPEED
DF81	CE, STOP
DF83	CF, SYMBOL
DF85	D0, TAG
DF86	D1, TAGOFF
DF89	D2, TROFF
DF8B	D3, TRON
DF8D	D4, WAIT
DF8F	D5, WEND
DF91	D6, WHILE
DF93	D7, WIDTH
DF95	D8, WINDOW
DF97	D9, WRITE
DF99	DA, ZONE
DF9B	DB, DI
DF9D	DC, EI
DF9E	DD, FILL
DFA1	DE, GRAPHICS
DFA3	DF, MASK
DFA5	E0, FRAME
DFA7	E1, CURSOR
DFAA	Beginn des freien RAMs
DFB2	max. 300 Zeichen
DFB5	Zeichen aus Eingabepuffer holen
DFB9	letztes Zeichen ?

DFBA	nein
DFBE	301 - Zählerstand
DFC0	gleich Zeilenlänge
DFC3	nach B
DFC6	dreimal Null als Abschluß
******	Zeichen aus Eingabepuffer holen
DFCD	
DFCE	letztes Zeichen ?
DFD0	Buchstabe ?
DFD3	ja
DFD5	numerisch ?
DFD8	ja
DFDB	'&' ?
DFDD	ja
DFE1	Token ?
DFE2	ja
DFE3	'!'
DFE8	zusätzliche Blanks ignorieren ?
DFEC	ja
DFED	' '
DFEF	in Puffer schreiben
DFF5	'REM' ?
DFF7	ja
DFFB	Basisadresse der Tabelle
DFFE	Tabelle durchsuchen
E002	gefunden, dann Rest nicht konvertieren
E005	'ELSE'
E009	in Puffer schreiben
E00D	Zeichen in Puffer schreiben
E00E	Pufferzeiger erhöhen
E00F	Zähler erniedrigen
E011	Zähler gleich Null ?
E012	nein
E013	Fehlermeldung ausgeben
E016	'Line too long'
******	spezielle Token
E017	'DATA'
E018	'DEFINT'
E019	'DEFSTR'

E01A	'DEFREAL'
E01B	Tabellenende
******	
E01C	in Puffer schreiben
E020	Pufferende ?
E022	':'
E026	Pufferzeiger erhöhen
E028	Token ?
E02B	' '
E02F	' '
E031	'"'
E03B	Flag für Ende des Statements
E044	Klein- in Großbuchstaben wandeln
E047	Adressen der Befehlsworte berechnen
E052	Test auf Buchstabe oder Ziffer
E058	'FN'
E05B	Test auf Buchstabe oder Ziffer
E069	'Funktion'
E06B	in Puffer schreiben
E06E	Funktionstoken
E06F	in Puffer schreiben
E07C	Test auf Buchstabe oder Ziffer
E086	Token für Variable
E08B	Token für Real-Variable
E090	in Puffer schreiben
E094	zweimal Null für Variablenadresse
E097	in Puffer schreiben
E09D	Test auf Buchstabe oder Ziffer
E0A3	in Puffer schreiben
E0B6	Basisadresse der Tabelle
E0B9	Tabelle durchsuchen
******	Befehle mit Zeilennummer
E0C8	'RESTORE'
E0C9	'AUTO'
E0CA	'RENUM'
E0CB	'DELETE'
E0CC	'EDIT'
E0CD	'RESUME'
E0CE	'ERL'

E0CF	'ELSE'
E0D0	'RUN'
E0D1	'LIST'
E0D2	'GOTO'
E0D3	'THEN'
E0D4	'GOSUB'
E0D5	Ende der Tabelle
E0D6	'!'
E0DA	'&'
E0DD	'$'
E0F9	Token für Zeilennummer
E105	Test auf String
E108	Token für Fließkommazahl
E112	Token für Zwei-Byte-Zahl
E119	10
E11D	Offset addieren
E121	Token für Ein-Byte-Zahl
E123	in Puffer schreiben
E128	in Puffer schreiben
E12F	in Puffer schreiben
E134	Vergleich HL <> DE
E145	Token für Binärzahl
E14B	in Puffer schreiben
E152	Variablentyp holen
E158	in Puffer schreiben
E161	'"'
E165	'|', Befehlserweiterung
E16B	'?'
E16D	Token für 'PRINT'
E172	Adresse der BASIC-Operatoren
E187	'''
E18B	in Puffer schreiben
E190	'"'
E19A	in Puffer schreiben
E1A1	'"'
E1A5	in Puffer schreiben
******	Befehlserweiterung verarbeiten
E1A8	in Puffer schreiben
E1AB	Null

E1AF	in Puffer schreiben
E1B2	nächstes Zeichen
E1B3	Zeiger erhöhen
E1B4	Test auf Buchstabe oder Ziffer
E1B7	ja, dann in Puffer
E1BA	Zeiger eins zurück
E1BC	beim letzten Zeichen Bit 7 setzen
E1C3	in Puffer schreiben
E1C6	’’’
E1C8	in Puffer schreiben
E1CB	Zeichen
E1CE	bis Zeilenende in Puffer schreiben
******	BASIC-Befehl LIST
E1D2	Zeilennummernbereich holen
E1D7	Kanalnummer holen
E1DA	Ende des Statements, sonst ’Syntax error’
E1DD	aktuelle Zeilenadresse auf Null
E1E2	Zeilen listen
E1E5	zum READY-Modus
******	BASIC-Zeilen BC- DE listen
E1E8	
E1E9	Zeilennummer nach DE
E1EB	BASIC-Zeile DE suchen
E1F0	Programmende ?
E1F5	fertig
E1F6	Unterbrechung durch ’ESC’
E1FA	Zeilenlänge addieren
E201	nächste Zeilennummer nach DE
E205	Vergleich HL <> DE
E209	größer letzte Zeilennummer ?
E20B	BASIC-Zeile in Puffer listen
E20E	Zeiger auf Puffer
E211	Zeichen ausgeben
E214	Zeiger erhöhen
E215	nächstes Zeichen
E217	noch nicht Ende ?
E219	LF ausgeben
E21F	nächste Zeile listen
E222	Ausgabekanal kleiner 8 ?

E225	Zeichen laden
E226	ja, Bildschirmausgabe
E228	Zeichen ausgeben
E22B	LF
E22E	CR
E232	Kontrollzeichen ?
E234	als druckbare Zeichen ausgeben
E23A	Zeichen ausgeben
E249	Zeiger auf Puffer
E26B	Befehlstoken ?
E27C	Konstante ausgeben
E27E	'l', Befehlserweiterung
E290	'"'
E294	'ELSE'
E299	':'
E29D	'"'
E2A8	'"'
E2CF	Zeichen in Puffer schreiben
E2D0	Pufferzeiger erhöhen
******	Befehlserweiterung listen
E2D6	
E2D8	in Puffer schreiben
E2DC	nächstes Zeichen
E2DD	Zeiger erhöhen
E2DE	Zeilenende ?
E2DF	nein
E2E1	Bit 7 löschen
E2E3	in Puffer schreiben
E2E6	letztes Zeichen ?
E2E8	nein, nächstes Zeichen
E2ED	' '
E2EF	in Puffer schreiben
E302	Funktion ?
E324	Test auf Buchstabe oder Ziffer
E337	Fließkommazahl ?
E33F	Binärzahl ?
E343	Hexzahl ?
E347	Zeilenadresse ?
E34B	Zeilennummer ?

E34F	Zwei-Byte-Zahl ?
E356	Ein-Byte-Zahl ?
E35B	Ziffer ?
E376	'X'
E38D	'&'
E398	Variablentyp 'Real'
E39A	Zahl holen
E3AE	'A'
E3B3	plus #E41D, Adresse der Befehlsworte
E3BF	26 Buchstaben
E3C1	Tabelle der Befehlsworte
E3CA	nächster Buchstabe
E3CC	Tabelle der BASIS-Operatoren
E3D2	'Syntax error'
E3F6	TAB
E3FA	' '
******	Adressen der Befehlsworte
E41D	A
E41F	B
E421	C
E423	D
E425	E
E427	F
E429	G
E42B	H
E42D	I
E42F	J
E431	K
E433	L
E435	M
E437	N
E439	O
E43B	P
E43D	Q
E43F	R
E441	S
E443	T
E445	U
E447	V

E449	W
E44B	X
E44D	Y
E44F	Z
******	Tabelle der BASIC-Befehle
******	Buchstabe Z
E451	DA ZONE
******	Buchstabe Y
E456	48 YPOS
******	Buchstabe X
E45B	47 XPOS
E45F	FD XOR
******	Buchstabe W
E463	D9 WRITE
E468	D8 WINDOW
E46E	D7 WIDTH
E473	D6 WHILE
E478	D5 WEND
E47C	D4 WAIT
******	Buchstabe V
E481	7F VPOS
E485	1D VAL
******	Buchstabe U
E489	ED USING
E48E	1C UPPER$
E494	1B UNT
******	Buchstabe T
E498	D3 TRON
E49C	D2 TROFF
E4A1	EC TO
E4A4	46 TIME
E4A7	EB THEN
E4AB	7D TESTR
E4B0	7C TEST
E4B4	1A TAN
E4B7	D1 TAGOFF
E4BD	D0 TAG
E4C0	EA TAB
******	Buchstabe S

E4C4	CF SYMBOL
E4CA	E7 SWAP
E4CE	7B STRING$
E4D5	19 STR$
E4D9	CE STOP
E4DD	E6 STEP
E4E1	18 SQR
E4E3	17 SQ
E4E6	CD SPEED
E4EB	E5 SPC
E4EE	16 SPACE$
E4F3	CC SOUND
E4F9	15 SIN
E4FC	14 SGN
E4FF	CB SAVE
******	Buchstabe R
E504	CA RUN
E507	7A ROUND
E50C	45 RND
E50F	79 RIGHT$
E515	C9 RETURN
E51B	C8 RESUME
E521	C7 RESTORE
E528	C6 RENUM
E52D	13 REMAIN
E533	C5 REM
E536	C4 RELEASE
E53D	C3 READ
E541	C2 RANDOMIZE
E549	C1 RAD
******	Buchstabe Q
******	Buchstabe P
E54F	BF PRINT
E554	78 POS
E557	BE POKE
E55B	BD PLOTR
E560	BC PLOT
E564	44 PI
E566	BB PEN

E569	12 PEEK
E56D	BA PAPER
******	Buchstabe O
E573	B9 OUT
E576	B8 ORIGIN
E57C	FC OR
E57E	B7 OPENOUT
E585	B6 OPENIN
E58B	B5 ON SQ
E598	B4 ON ERROR GOTO 0
E5A0	B3 ON BREAK
E5A8	B2 ON
******	Buchstabe N
E5AB	FE NOT
E5AE	B1 NEW
E5B1	B0 NEXT
******	Buchstabe M
E5B6	AF MOVER
E5BB	AE MOVE
E5BF	AD MODE
E5C3	FB MOD
E5C6	77 MIN
E5C9	AC MID$
E5CD	AB MERGE
E5D2	AA MEMORY
E5D8	76 MAX
E5DB	DF MASK
******	Buchstabe L
E5E0	11 LOWER$
E5E6	10 LOG10
E5EB	0F LOG
E5EE	A9 LOCATE
E5F4	A8 LOAD
E5F8	A7 LIST
E5FC	A6 LINE
E600	A5 LET
E603	0E LEN
E606	75 LEFT$
******	Buchstabe K

E60C	A4 KEY
******	Buchstabe J
E610	OD JOY
******	Buchstabe I
E614	0C INT
E617	74 INSTR
E61C	A3 INPUT
E621	0B INP
E624	43 INKEY$
E62A	0A INKEY
E62F	A2 INK
E632	A1 IF
******	Buchstabe H
E635	42 HIMEM
E63A	73 HEX$
******	Buchstabe G
E63F	DE GRAPHICS
E647	A0 GO TO
E64C	9F GO SUB
******	Buchstabe F
E653	09 FRE
E656	E0 FRAME
E65B	9E FOR
E65E	E4 FN
E660	08 FIX
E663	DD FILL
******	Buchstabe E
E668	07 EXP
E66B	9D EVERY
E670	9C ERROR
E675	41 ERR
E678	E3 ERL
E67B	9B ERASE
E680	40 EOF
E683	9A ENV
E686	99 ENT
E689	98 END
E68C	97 ELSE
E690	DC EI

E692	96 EDIT
******	Buchstabe D
E697	95 DRAWR
E69C	94 DRAW
E6A0	93 DIM
E6A3	DB DI
E6A5	49 DERR
E6A9	92 DELETE
E6AF	91 DEG
E6B2	90 DEFSTR
E6B8	8F DEFREAL
E6BF	8E DEFINT
E6C5	8D DEF
E6C8	72 DEC$
E6CC	8C DATA
******	Buchstabe C
E6D1	E1 CURSOR
E6D7	06 CREAL
E6DC	05 COS
E6DF	7E COPYCHR$
E6E7	8B CONT
E6EB	8A CLS
E6EE	89 CLOSEOUT
E6F6	88 CLOSEIN
E6FD	87 CLG
E700	86 CLEAR
E705	04 CINT
E709	03 CHR$
E70D	85 CHAIN
E712	84 CAT
E715	83 CALL
******	Buchstabe B
E71A	82 BORDER
E720	71 BIN$
******	Buchstabe A
E725	81 AUTO
E729	02 ATN
E72C	01 ASC
E72F	FA AND

E732	80 AFTER
E737	00 ABS
******	BASIC-Operatoren und zugehörige Token
E73B	F8 ’^’
E73D	F9 ’Backslash’
E740	F0 ’>=’
E743	F0 ’=>’
E747	EE ’>’
E749	F2 ’<>’
E74D	F3 ’<=’
E751	F3 ’=<’
E755	EF ’=’
E757	F1 ’<’
E759	F7 ’/’
E75B	01 ’:’
E75D	F6 ’*’
E75F	F5 ’-’
E761	F4 ’+’
E763	C0 ’’’
******	Programmzeiger löschen
E766	
E767	Programmstart
E76D	dreimal Null ans Programmende
E770	Programmende
E77D	Zeilennummer einsetzen
******	Zeilenadressen durch Zeilennummer ersetzen
E78B	nächstes Element der Zeile holen
E78E	Ende des Statements ?
E790	ja
E791	’Zeilenadresse’ ?
E793	nein
E79F	Zeilennummer nach DE
E7A2	’Zeilennummer’
E7A6	einsetzen
******	BASIC-Befehl DELETE
E7F3	
E7F6	Ende des Statements, sonst ’Syntax error’
E802	zum READY-Modus
E805	Zeilennummernbereich holen

E80A	BASIC-Zeile DE suchen
E80F	BASIC-Zeile DE suchen
******	Zeilenadresse holen
E82C	
E82E	Nummer oder Adresse nach DE
E830	'Zeilenadresse' ?
E832	ja
E834	'Zeilennummer' ?
E836	'Syntax error'
E83A	Zeilennummer nach HL holen
E83D	Vergleich HL <> DE
E840	kleiner, ab Programmstart suchen
E845	Rest der Zeile überlesen
E848	ab Adresse (HL)
E849	BASIC-Zeile DE suchen
E84C	nicht gefunden, ab Programmstart suchen
E854	'Zeilenadresse'
E859	ins Programm
E85B	Zeilenadresse ins Programm
******	BASIC-Zeile DE suchen
E861	
E865	Fehlermeldung ausgeben
E868	'Line does not exist'
E869	Programmstart
E86E	Zeilenlänge nach BC
E872	Programmende ?
E873	nicht gefunden
E878	Zeilennummer nach HL
E87C	Vergleich HL <> DE
E882	größer, nicht gefunden
E883	gleich, gefunden
E884	Zeilenlänge addieren
E885	weitersuchen
******	BASIC-Zeile DE suchen
E887	Programmstart
E88B	Zeilenadresse merken
E88D	Zeilenlänge nach BC
E891	Programmende ?
E893	ja

E896	Zeilennummer nach HL
E89A	Vergleich HL <> DE
E89F	laufende Zeilennummer größer oder gleich ?
E8A0	Zeilenlänge addieren
E8A1	weitersuchen
******	BASIC-Befehl RENUM
E8A3	10, Default für Startwert
E8A6	Zeilennummer nach DE holen
E8AA	0, Default
E8AD	folgt Komma ?
E8B0	Zeilennummer nach DE holen
E8B4	10, Default
E8B7	folgt Komma ?
E8BD	Zeilenende, sonst 'Syntax error'
E8C6	BASIC-Zeile DE suchen
E8CB	BASIC-Zeile DE suchen
E8D0	Vergleich HL <> DE
E8D3	'Improper argument'
E8F2	'Improper argument'
E967	'IF'
E974	'ELSE'
E980	'['
E984	'('
E98D	'['
E991	'('
E995	']'
E999	')'
E9A1	'Syntax error'
******	BASIC-Befehl DATA
E9A8	
******	BASIC-Befehle REM und '
E9AC	
******	BASIC-Befehl ELSE
E9B2	
E9C2	Programmstart
E9D1	Sprung nach (BC)
E9EC	Fehlermeldung ausgeben
E9FA	'ELSE'
E9FD	'THEN'

EA02	Blanks überlesen
EA0E	’"’
EA12	’I’
EA16	’’’
EA1A	’REM’
EA1E	Funktion
EA25	’"’
EA2C	’"’
******	BASIC-Befehl RUN
EA7D	Ende des Statements ?
EA80	Programmstart als Default
EA84	ja
EA86	Zeilennummer ?
EA88	ja
EA8A	Zeilenadresse ?
EA94	MC BOOT PROGRAMM
EA9A	Programmstart
EA9F	Zeilenadresse holen
EAB7	zur Interpreterschleife
******	BASIC-Befehl LOAD
EABA	
EAC2	zum READY-Modus
EACC	DISK IN DIRECT
EAD6	Namen holen, File öffnen
EAD9	Filetyp
EAE7	folgt Komma ?
EAEA	ja, 16-Bit-Wert holen
EAED	als Startadresse
EAF1	Ende des Statements, sonst ’Syntax error’
EAF6	Startadresse
EAF9	DISK IN DIRECT
EAFD	DISK IN CLOSE
******	BASIC-Befehl CHAIN
EB02	’MERGE’
EB04	Flag für MERGE
EB07	Blanks überlesen
EB0D	Defaultwert Null für Startzeile
EB10	folgt Komma ?
EB13	nein

EB16	','
EB18	16-Bit-Wert holen
EB1C	folgt Komma ?
EB1F	nein
EB21	Test auf nachfolgendes Zeichen
EB24	'DELETE'
EB25	Zeilenbereich löschen
EB2A	Ende des Statements, sonst 'Syntax error'
EB30	Garbage Collection
EB3D	Startzeile holen
EB3E	Programmstart als Default
EB42	keine Startzeile
EB4C	Flag für MERGE
******	BASIC-Befehl MERGE
EB59	Namen holen, File öffnen
EB5C	Ende des Statements, sonst 'Syntax error'
EB5F	Variablen löschen
EB62	Filetyp testen
EB65	zum READY-Modus
EB71	Programmende
EB75	Programmstart
EB79	Programmende
EB7D	BD := HL - DE
EBA5	Vergleich HL <> DE
EBBA	Programmende
EBCD	Vergleich HL <> DE
EBE9	Programmende
EBF1	Programmende
EBFD	'Memory full'
EBFF	Fehlermeldung ausgeben
EC01	DISK IN CHAR
EC05	CTRL Z
EC09	Diskettenfehler
EC0E	Diskettenfehler
EC14	'EOF met'
EC1B	Fehlermeldung ausgeben
EC24	Programmende
EC2A	Programmende
EC32	Programmende

EC4B	Programmende
EC67	Filetyp
EC6E	ASCII-File ?
EC70	nein
EC72	Filetyp
EC75	ASCII-File
EC77	ja
EC79	Bit 0 (geschütztes File) löschen
EC7D	Fehlermeldung ausgeben
EC80	'File type error'
EC87	Programmstart
EC9A	Vergleich HL <> DE
ECA2	Programmende
ECA5	Filetyp
ECA8	Bit 0 testen
ECAA	Flag für geschütztes File setzen
ECAF	DISK IN DIRECT
ECB2	'EOF met'
ECD8	'Direct command found'
ECDC	'Overflow'
ECDE	Fehlermeldung ausgeben
******	BASIC-Befehl SAVE
ECE1	
ECE4	OPENOUT
ECE7	Filetyp 0, BASIC-Programm
ECE9	folgt Komma ?
ECEC	nein
ECEE	Test auf nachfolgendes Zeichen
ECF1	normale Variable
ECF4	Variablenname
ECF5	Klein- in Großbuchstaben wandeln
ECF7	'Syntax error'
ECFB	Basisadresse der Tabelle
ECFE	Tabelle durchsuchen
ED01	Adresse aus Tabelle auf Stack
ED02	Blanks überlesen
ED05	Anzahl der Einträge
ED06	nicht gefunden, 'Syntax error'
ED08	'A'

ED0B	'B'
ED0E	'P'
******	SAVE ,P
ED11	Filetyp 1, protected
ED13	Ende des Statements, sonst 'Syntax error'
ED1E	Programmstart
ED23	Programmende
ED27	HL := HL - DE
******	SAVE ,B
ED30	Test auf ','
ED33	16-Bit-Wert holen
ED36	merken
ED37	Test auf ','
ED3A	16-Bit-Wert holen
ED3D	merken
ED3E	folgt Komma ?
ED41	0, Default für Einsprungadresse
ED44	ja, 16-Bit-Wert holen
ED47	merken
ED48	Ende des Statements, sonst 'Syntax error'
ED4B	Filetyp 2, binär
ED4D	Einsprungadresse
ED4E	Endadresse
ED4F	Startadresse
ED50	DISK OUT DIRECT
ED53	Unterbrechung durch 'ESC'
ED56	CLOSEOUT
******	SAVE ,A
ED58	Ende des Statements, sonst 'Syntax error'
ED5C	9
ED5E	Ausgabe auf Kanal 9, Diskette
ED62	1 bis
ED65	65535
ED68	Zeilen listen
ED6C	Ausgabe wieder auf Default
ED6F	CLOSEOUT
ED79	Blank, TAB und LF löschen
ED7E	'&'
ED82	Test auf numerisch

ED87	Typ auf Integer
ED8A	Variable löschen
ED9A	'&'
EDA3	Integerzahl in HL übernehmen
EDB0	'.'
EDB2	Blank, TAB und LF löschen
EDB5	Test auf Ziffer
EDBC	'.'
EDC1	Typ auf Integer
EDCE	'.'
EDD5	Typ auf Real
EDEF	Test auf String
EDF2	nein
EDFF	4-Byte Integer*256 nach Fließkomma
EE03	Zahl mit 10^A multiplizieren
EE07	Variablentyp auf Fließkomma setzen
EE0A	Variable von (DE) nach (HL) kopieren
EE14	Blank, TAB und LF löschen
EE18	-1
EE1A	'-'
EE1D	0
EE1E	'+'
EE24	Blank, TAB und LF löschen
EE28	Test auf Ziffer
EE2E	Klein- in Großbuchstaben wandeln
EE33	'0'
EE47	'E'
EE52	Blank, TAB und LF löschen
EE55	Test auf Ziffer
EE60	Typ auf Real setzen
EE6B	100
EE99	Blank, TAB und LF überlesen
EEC0	mal 10
EEC4	plus nächste Ziffer
EEDA	Integerzahl in HL übernehmen
EEED	Blank, TAB und LF überlesen
EEF0	Klein- in Großbuchstaben wandeln
EEF3	Basis 2, binär
EEF5	'X'

EEF9	Basis 16, hex
EEFB	'H'
EF00	Blank, TAB und LF überlesen
EF05	Basis 10, dezimal
EF08	(Hex-) Ziffer nach binär wandeln
EF12	(Hex-) Ziffer nach binär wandeln
EF1C	Basis des Zahlensystems
EF1D	Integermultiplikation ohne Vorzeichen
******	(Hex-) Ziffer nach binär wandeln
EF31	Zeichen holen
EF32	Zeiger erhöhen
EF33	Test auf Ziffer
EF36	ja
EF38	Klein- in Großbuchstaben wandeln
EF3B	'A'
EF3E	kleiner 'A', Fehler
EF40	'A'-('9'+1)
EF42	'0'
EF45	kein Fehler
EF47	Carry löschen
******	Integerzahl HL ausgeben
EF49	Integerzahl nach ASCII wandeln
EF4C	String ausgeben
******	Integerzahl nach ASCII wandeln
EF4F	
EF51	Integerzahl in HL übernehmen
EF61	Null
EF62	Zahl in formatierten String wandeln
EF68	' '
EF98	'%'
F02F	'.'
F034	'1'
F03D	'+E'
F047	'-'
F04C	'0'-1
F04F	10
F053	'9'+1
F079	'.'
F099	'0'

F0A8	'5'
F0B1	'1'
F0C0	'9'
F0C3	'0'
F0DA	'0'
F0E8	'0'
F128	','
F135	'0'
F146	'0'
F156	'-'
F162	'+'
F166	' '
F181	'*'
F185	' '
F1CF	'0'
F1DE	'0'
******	BASIC-Funktion PEEK
F20D	UNT
F210	READ RAM, LD A,(HL)
F211	Akkuinhalt als Integerzahl übernehmen
******	BASIC-Befehl POKE
F214	16-Bit-Adresse holen
F218	Komma testen und 8-Bit-Wert holen
F21C	Wert in Adresse schreiben
******	BASIC-Funktion INP
F21E	CINT
F221	Portadresse nach BC
F223	Port lesen
F225	Akkuinhalt als Integerzahl übernehmen
******	BASIC-Befehl OUT
F228	Adresse und Wert holen
F22B	ausgeben
******	BASIC-Befehl WAIT
F22E	Adresse und Wert holen
F231	8-Bit-Wert nach D
F232	3. Parameter Null
F234	evtl. dritten Parameter holen
F237	dritter Parameter nach E
F238	Port lesen

F23B	verknüpfen
F23C	und warten
******	16-Bit- und 8-Bit-Wert holen
F23F	16-Bit-Wert holen
F243	nach BC
F244	Test auf ','
F247	und 8-Bit-Wert holen
******	Befehlserweiterung
F24A	Programmzeiger erhöhen
F24C	folgt Nullbyte ?
F24F	ja, KL FIND COMMAND
F252	Befehlsadresse nach DE
F254	nicht gefunden, 'Unknown command'
F256	Zeichen holen
F257	Befehlswort überlesen
F258	Bit 7 gesetzt ?
F259	nein, weiterlesen
F25B	zum CALL-Befehl
F25D	Fehlermeldung ausgeben
F260	'Unknown command'
******	BASIC-Befehl CALL
F261	16-Bit-Wert holen
F264	#FF = RAM selektiert
F266	Adresse nach #AE55
F26B	Konfigurationsbyte nach #AE57
F26E	Stackpointer retten
F272	maximal 32 Parameter
F274	folgt Komma ?
F277	nein
F27A	Ausdruck holen
F27E	und Adresse auf Stack
F27F	nächsten Parameter
F281	Ende des Statements, sonst 'Syntax error'
F284	HL retten
F289	Anzahl der Parameter in Akku
F28E	Adresse des Parameterblocks nach IX
F290	Routine ausführen
F293	Stackpointer zurück
F297	Descriptorstack initialisieren

F29A	HL zurück
******	BASIC-Befehl ZONE
F2A2	8-Bit-Wert ungleich Null holen
F2A5	als Tabulatorweite
******	BASIC-Befehl PRINT
F2A9	Kanalnummer
F2AC	Ende des Statements ?
F2AF	ja, LF ausgeben
F2B2	'USING'
F2B8	Basisadresse der Tabelle
F2BB	Tabelle durchsuchen
F2BF	JP (DE), Funktion ausführen
F2C2	Ende des Statements ?
F2C5	nein, weitermachen
F2C8	Anzahl der Tabelleneinträge
F2C9	Rücksprungadresse falls nicht gefunden
F2CB	','
F2CE	'SPC'
F2D1	'TAB'
F2D4	';'
F2D5	Blanks überelesen
******	PRINT
F2D7	Ausdruck holen
F2DC	Test auf String
F2DF	ja
F2E1	Zahl in ASCII-String wandeln
F2E4	Stringparameter holen
F2E7	' ' Leerzeichen anhängen
F2E9	Stringdescriptor holen
F2EC	Länge erhöhen
F2F0	Stringdescriptor holen
F2F3	Länge
F2FF	' '
F302	Kontrollzeichen ?
F30F	Ausgabestrom selektieren
******	PRINT ,
F31E	Blanks überlesen
F321	Tabulatorweite
******	PRINT SPC

F339	8-Bit-Wert in Klammern holen
******	PRINT TAB
F342	8-Bit-Wert in Klammern holen
******	8-Bit-Wert in Klammern holen
F362	Blanks überlesen
F365	Test auf '('
F368	8-Bit-Wert holen
F36B	Test auf ')'
******	PRINT USING
F383	Blanks überlesen
F386	Stringausdruck holen
F389	Test auf nachfolgendes Zeichen
F38C	';'
F392	Ausdruck holen
F3A9	Ende des Statements ?
F3AC	ja
F3B4	Ausdruck holen
F3D7	'Underline'
F3F4	';'
F3F6	Blanks überlesen
F3F9	Test auf ','
F3FF	'&'
F404	'!'
F408	'Backslash'
F413	'Backslash'
F417	' '
F436	auf Formatierungszeichen prüfen
F443	Zahl formatieren
F446	String ausgeben
******	auf Formatierungszeichen prüfen
F44D	
F454	'+'
F460	'.'
F464	'#'
F47A	' '
F47C	'*'
F489	'#'
F49C	'$'
F4B0	'Improper argument'

F4B8	'.'
F4BC	'#'
F4C0	','
F4D0	'#'
F4D6	'^'
F4F9	'-'
F4FD	'+'
F507	'$'
******	BASIC-Befehl WRITE
F50D	
F510	Ende des Statements
F513	ja
F516	Ausdruck holen
F51B	Test auf String
F51E	ja
F520	Zahl nach ASCII wandeln
F523	und ausgeben
F528	'"'
F52A	ausgeben
F52D	String ausgeben
F530	'"'
F532	ausgeben
F537	Ende des Statements ?
F53D	','
F53F	ausgeben
F542	weitermachen
******	Speicher konfigurieren
F544	Speicherplatz von DE bis HL
F547	Vergleich HL mit BC
F54A	höchstes Adresse < #AC00 ?
F54B	HIMEM
F54E	Ende der Strings
F551	Ende des freien RAMs
F555	Beginn des freien RAMs
F558	plus 303
F55D	ergibt Programmstart
******	BASIC-Befehl MEMORY
F570	16-Bit-Wert holen
F577	Vergleich HL <> DE

******	
F58F	TXT GET M TABLE
F5F7	Vergleich HL <> DE
******	Länge des Stringbereichs berechnen
F5FD	
F5FF	Beginn der Strings
F603	Ende der Strings
F606	BC := HL - DE
******	Prg- und Variablenzeiger um BC erhöhen
F60C	Programmende
F610	Programmende
F618	Variablenstart
F61C	Variablenstart
F61F	Arraystart
F623	Arraystart
F626	Arrayende
F62A	Arrayende
F633	Programmende
F63E	Programmende
F645	BC := HL - DE
******	BASIC-Stack initialisieren
F652	Beginn des Stacks
F655	BASIC-Stackpointer
F658	für ein Byte
F65A	Platz im BASIC-Stack reservieren
F65D	Null auf Stack
F65F	Stackpointer erhöhen
F660	und merken
******	Platz im BASIC-Stack freigeben
F665	Stackpointer
F669	Akkuinhalt abziehen
F671	neuen Wert des BASIC-Stackpointers merken
******	Platz im BASIC-Stack reservieren
F675	BASIC-Stackpointer
F67A	Akkuinhalt addieren
F67E	BASIC-Stackpointer
F683	ergibt plus #4F94 Überlauf ?
F686	dann ist Stackpointer > #B06C
F689	BASIC-Stack initialisieren

F68C	'Memory full'
F68F	Ende der Strings
F692	Beginn der Strings
******	Platz für String reservieren
F696	
F69C	Beginn der Strings
F6A4	Vergleich HL <> DE
F6AE	Fehlermeldung ausgeben
F6B1	'String space full'
F6B2	Beginn der Strings
F6BF	Programmende
F6D2	Vergleich HL <> DE
F6EA	Blocktransfer LDDR
F6F9	BC := HL - DE
F6FE	Blocktransfer LDIR
F705	BC := HL - D
F70C	Beginn der Strings
F717	Beginn der Strings
******	BASIC-Befehl SYMBOL
F784	'AFTER'
F788	8-Bit-Wert holen
F78C	Test auf ','
F78F	8 Werte
F792	folgt Komma ?
F796	ja, 8-Bit-Wert holen
F79B	schon 8 Argumente ?
F79F	TXT GET MATRIX
F7A2	Matrix nicht im RAM, 'Improper argument'
F7A5	8
F7A8	plus Matrixadresse
F7A9	Byte vom Stack
F7AB	in Matrixtabelle
F7AD	nächstes Byte
******	SYMBOL AFTER
F7B1	Blanks überlesen
F7B4	Integerwert mit Vorzeichen holen
F7B8	256
F7BB	Vergleich HL <> DE
F7BE	'Improper argument'

F7C2	TXT GET M TABLE
F7C6	Matrix noch nicht definiert ?
F7D3	'Improper argument'
F7DD	256
F7E0	TXT SET M TABLE
F7FD	'Memory full'
F805	TXT SET M TABLE
F815	Vergleich HL <> DE
F818	'Memory full'
F833	Ende der Strings
F83E	Blocktransfer LDDR
F844	Beginn der Strings
F851	Blocktransfer LDIR
F857	Ende der Strings
F85B	Beginn der Strings
F865	Programmende
F868	Vergleich HL <> DE
F875	Fehlermeldung ausgeben
F878	'Memory full'
******	String lesen
F879	
F87E	'"'
F880	Blanks überlesen
F89F	','
F8AD	JP (DE)
F8BE	' '
F8C2	TAB
F8C6	CR
F8CA	LF
******	String ausgeben
F8D0	Stringparameter holen
F8D3	Leerstring ?
F8D4	Zeichen holen
F8D5	Zeiger erhöhen
F8D6	Zeichen ausgeben
F8D9	nächstes Zeichen
******	BASIC-Funktion LOWER$
F8EC	Groß- in Kleinbuchstaben wandeln
******	Groß- in Kleinbuchstaben wandeln

F8F1	'A'
F8F4	'Z'+1
F8F7	'a'-'A'
******	BASIC-Funktion UPPER$
F8FA	Klein- in Großbuchstaben wandeln
F915	Sprung nach (BC)
******	Stringaddition
F91D	Zeiger auf zweiten String
F921	Längen
F922	addieren
F923	kein]berlauf
F925	Fehlermeldung ausgeben
F928	'String too long'
******	BASIC-Funktion BIN$
F964	
******	BASIC-Funktion HEX$
F969	
F96D	Ausdruck holen
F975	folgt Komma ?
F978	0 als Default
F979	ja, 8-Bit-Wert holen
F97C	größer gleich 17 ?
F97E	ja, 'Improper argument'
F982	Test auf ')'
F98A	Zahl in String umwandeln
******	BASIC-Funktion DEC$
F98F	Ausdruck holen
F992	Test auf ','
F995	auf BASIC-Stack ablegen
F998	Stringausdruck und Parameter holen
F99B	Test auf ')'
F99F	Länge
F9A0	Platz im BASIC-Stack freigeben
F9A4	Länge
F9A5	Variable übernehmen
F9AB	auf Formatierungszeichen prüfen
F9AE	'Improper argument'
F9B3	'Improper argument'
F9B7	Zahl formatieren

F9BA	String übernehmen
******	BASIC-Funktion STR$
F9BC	
F9BD	Zahl in String wandeln
F9C1	Zähler für Stringlänge auf -1
F9C3	Null
F9C4	Zähler erhöhen
F9C5	Ende des Strings ?
F9C6	Zeiger erhöhen
F9C7	nein, nächstes Zeichen
F9CA	Stringlänge
F9CB	Platz reservieren, Stringdescriptor anlegen
******	BASIC-Funktion LEFT$
F9D3	String und 8-Bit-Wert holen
******	BASIC-Funktion RIGHT$
F9D8	String und 8-Bit-Wert holen
F9DB	Stringlänge
F9DC	minus Parameter
******	BASIC-Funktion MID$
F9E2	Test auf '('
F9E5	String und 8-Bit-Wert holen
F9E8	Null, 'Improper argument'
F9EB	255
F9EC	als Default
F9ED	drittes Argument holen
F9F0	Test auf ')'
******	BASIC-Befehl MID$
FA07	Test auf '('
FA0A	Variable holen
FA0D	Typ String, sonst 'Type mismatch'
FA19	'Improper argument'
FA1C	255
FA1D	als Default
FA1E	drittes Argument holen
FA21	Test auf ')'
FA24	Test auf '='
FA28	Stringausdruck und -Parameter holen
FA3E	Blocktransfer LDIR
FA43	Stringausdruck holen

******	drittes Argument für MID$ holen
FA4F	Default 255
FA52	')'
FA56	Test auf ','
FA59	8-Bit-Wert holen
******	BASIC-Funktion LEN
FA69	Stringparameter holen, Länge nach A
FA6C	Akkuinhalt als Integerzahl übernehmen
******	BASIC-Funktion ASC
FA6E	ASCII-Kode des ersten Zeichens
FA71	Akkuinhalt als Integerzahl übernehmen
******	BASIC-Funktion CHR$
FA74	CINT, <256
FA77	ASCII-Kode im Akku
FA7A	Länge 1
FA7C	String mit Länge A anlegen
******	BASIC-Funktion INKEY$
FA7E	KM READ CHR
FA81	keine Taste gedrückt ?
FA83	'ESC'
FA85	Leerstring
FA87	'ESC
FA89	Leerstring
FA8B	Zeichen in String übernehmen
******	BASIC-Funktion STRING$
FA8D	8-Bit-Wert holen, Länge
FA91	Test auf ','
FA94	Ausdruck holen
FA97	Test auf ')'
FA9F	String mit Länge A anlegen
FAA1	Test auf String
FAA4	nein
FAA6	Stringparameter holen
FAA9	Leerstring, 'Improper argument'
FAAB	ASCII-Kode holen
******	BASIC-Funktion SPACE$
FAAD	
FAB0	' '
******	BASIC-Funktion VAL

FABE	Stringparameter holen
FAC1	Akkuinhalt als Integerzahl übernehmen
FACE	String in Zahl umwandeln
FAD5	Fehlermeldung ausgeben
FAD8	'Type mismatch'
FAE2	'Improper argument'
******	BASIC-Funktion INSTR
FAE5	Ausdruck holen
FAE8	Test auf String
FAEB	Default Startposition 1
FAED	ja
FAEF	CINT, < 256
FAF3	'Improper argument'
FAF7	Test auf ','
FAFA	Stringausdruck holen
FAFD	Test auf ','
FB05	Stringausdruck und -Parameter holen
FB08	Test auf ')'
FB48	Akkuinhalt als Integerzahl übernehmen
FB65	Beginn der Strings
FB68	Vergleich HL <> BC
FB6D	Ende der Strings
FB70	Vergleich HL <> BC
FB9E	Programmstart
FBA1	Vergleich HL <> DE
FBA4	Ende der Strings
FBA8	Vergleich HL <> DE
FBAD	Programmende
FBB1	Vergleich HL <> DE
FBC4	Blocktransfer LDIR
******	Descriptorstack initialisieren
FBCC	
FBCF	Zeiger in Descriptorstack für Strings
FBD7	String
FBD9	als Variablentyp
FBDC	Zeiger in Descriptorstack
FBE2	Stringdescriptor
FBE5	Vergleich HL <> DE
FBE8	'String expression too complex'

FBEA	Fehlermeldung ausgeben
FBF0	Zeiger in Descriptorstack
FBF6	Typ String, sonst 'type mismatch'
FC0A	Beginn der Strings
FC10	Vergleich HL <> DE
FC1B	Beginn der Strings
FC27	Vergleich HL <> DE
******	BASIC-Funktion FRE
FC53	Test auf String
FC56	nein
FC5B	Garbage Collection
FC5E	freien Speicherplatz berechnen
******	Garbage Collection
FC64	
FC7C	Vergleich HL <> DE
FC87	Ende der Strings
FC8B	Beginn der Strings
FCA9	Vergleich HL <> BC
FCAF	Beginn der Strings
FCB3	BC := HL - DE
FCB7	Vergleich HL <> DE
FCBC	Blocktransfer LDDR
FCC0	Beginn der Strings
FCD9	Vergleich HL <> DE
FCE6	Vergleich HL <> DE
FCF3	numerisches Ergebnis holen
FD03	UNT
******	BASIC-Operator '+'
FD0C	Typ der Operanden testen
FD0F	Fließkomma ?
FD11	Integer-Addition HL := HL + DE
FD14	kein]berlauf, Ergebnis in HL übernehmen
FD17	nach Fließkomma wandeln
FD1A	Fließkomma-Addition
FD1D	kein]berlauf, ok
FD1E	'Overflow'
******	BASIC-Operator '-'
FD21	Typ der Operanden testen
FD24	Fließkomma ?

FD26	Integer-Subtraktion HL := DE - HL
FD29	kein]berlauf, Ergebnis in HL übernehmen
FD2C	nach Fließkomma wandeln
FD2F	Fließkomma-Subtraktion
FD32	kein]berlauf, ok
FD33	'Overflow'
******	BASIC-Operator '*'
FD35	Typ der Operanden testen
FD38	Fließkomma ?
FD3A	Integer-Multiplikation mit Vorzeichen
FD3D	kein]berlauf, Ergebnis in HL übernehmen
FD40	nach Fließkomma wandeln
FD43	Fließkomma-Multiplikation
FD46	kein]berlauf, ok
FD47	'Overflow'
******	arithmetischer Vergleich
FD49	Typ der Operanden testen
FD4C	Integer-Vergleich
FD4F	Fließkomma-Vergleich
******	BASIC-Operator '/'
FD52	
FD57	Fließkomma-Division
FD5B	5 Bytes
FD5E	Ergebnis übertragen
FD60	ok ?
FD61	'Division by zero'
FD64	'Overflow'
******	BASIC-Operator 'Backslash'
FD67	
FD6B	Integer-Division mit Vorzeichen
FD6E	Ergebnis in HL übernehmen
FD71	'Division by zero'
******	BASIC-Operator 'MOD'
FD79	
FD7D	MOD-Berechnung
FD80	Ergebnis in HL übernehmen
FD83	Fehlermeldung ausgeben
FD86	'Division by zero'
******	BASIC-Operator 'AND'

FD87	
FD8C	HL := HL AND DE
FD8F	Integerzahl HL übernehmen
******	BASIC-Operator 'OR'
FD92	
FD97	HL := HL OR DE
FD9A	Integerzahl HL übernehmen
******	BASIC-Operator 'XOR'
FD9C	
FDA1	HL := HL XOR DE
FDA4	Integerzahl HL übernehmen
******	BASIC-Operator 'NOT'
FDA6	CINT
FDAC	HL komplementieren
FDAE	Integerzahl HL übernehmen
******	BASIC-Funktion ABS
FDB0	SGN
FDB3	positives Vorzeichen, fertig
******	Vorzeichen umkehren
FDB4	numerisches Ergebnis holen
FDB7	Vorzeichenwechsel Fließkomma
FDBA	Vorzeichenwechsel Integer
FDBD	Ergebnis speichern
FDC0	]berlauf, Zahl nach Fließkomma wandeln
******	Vorzeichen bestimmen
FDC4	
FDC6	SGN
******	Vorzeichen bestimmen
FDCC	numerisches Ergebnis holen
FDCF	Integer SGN
FDD2	SGN
******	Zahl runden
FDD5	
FDD7	Variablentyp und -wert übernehmen
FDDB	numerisches Ergebnis holen
FDDE	Rundungsstellen
FDDF	Fließkommawert ?
FDE2	Rundung hinter Komma? fertig
FDE3	Integerwert nach Fließkomma wandeln

FDE6	Zahl runden
FDE9	CINT
FDEC	Rundungststellen
FDED	ungleich Null, dann runden
FDEF	Fließkomma nach Integer wandeln
FDF2	Funktion ausführen
FDF6	Rundungsstellen
FDF7	Fließkommazahl mit 10^A multiplizieren
FDFA	Fließkomma nach Integer wandeln
FE02	Integer nach Fließkomma wandeln
FE05	Rundungsstellen invertieren
FE06	entspricht Division
FE07	Fließkommazahl mit 10^A multiplizieren
******	BASIC-Funktion FIX
FE0E	FIX-Funktion
******	BASIC-Funktion INT
FE13	INT-Funktion
FE16	numerisches Ergebnis holen
FE19	Integer ?
FE1A	JP (DE), Funktion ausführen
FE1E	Variablentyp
FE25	Variablentyp nach C, Zeiger nach HL
FE29	Integer nach Fließkomma wandeln
FE2D	String ?
FE34	falls positiv Vorzeichen von B übernehmen
FE38	Ergebnis in HL übernehmen
FE3C	String ?
FE3E	ja, 'Type mismatch'
FE40	Variablentyp
FE43	String ?
FE45	ja, 'Type mismatch'
FE4D	Variablentyp
FE50	merken
FE52	Integerzahl nach Fließkomma wandeln
FE58	Integerzahl nach Fließkomma wandeln
FE5D	Zeiger auf Variable
FE68	Variable
FE6D	'Type mismatch'
FE70	Variablentyp

FE73	vergleichen
FE74	Integer ?
FE76	nein
******	Integer-Operanden nach Fließkomma
FE78	ersten Operand
FE7B	umwandeln
FE7E	BASIC-Stackpointer, zweiter Operand
FE81	umwandeln
FE84	nach DE
******	Integerzahl nach Fließkomma wandeln
FE8D	Zahl nach DE
******	Integerzahl nach Fließkomma wandeln
FE95	Variablentyp
FE98	auf 'Real'
FE9E	negativ, dann Integer-Vorzeichenwechsel
FEA2	Integer nach Fließkomma wandeln
******	4-Byte-Zahl nach Fließkomma wandeln
FEA5	Lo-Word
FEA9	Hi-Word
FEAC	Variablentyp
FEAF	'Real'
FEB1	Zeiger auf 4-Byte-Wert
FEB3	Zahl nach Fließkomma wandeln
******	BASIC-Funktion CINT
FEB6	
FEBA	'Overflow'
FEBF	Ergebnis
FECC	'Overflow'
FECE	Zeiger auf Variablentyp
FED1	Variablentyp laden
FED2	Typ auf Integer
FED5	mit String vergleichen
FED9	'Type mismatch'
FEDD	Fließkommazahl nach Integer konvertieren
FEE1	Vorzeichen B in Integerzahl HL übernehmen
******	Integerwert (HL) nach HL
FEE6	
******	BASIC-Funktion UNT
FEEB	numerisches Ergebnis holen

FEEE	Integer ?
FEEF	Fließkomma nach Integer umwandeln
FEF2	'Overflow'
FEF5	Vorzeichen B in Integerzahl übernehmen
FEF8	Integerzahl in HL übernehmen
FEFB	Fehlermeldung ausgeben
FEFE	'Overflow'
FF02	Variablentyp
FF05	vergleichen
FF06	unterschiedlich ?
FF0F	CINT
FF11	Typ String, sonst 'Type mismatch'
******	BASIC-Funktion CREAL
FF14	numerisches Ergebnis holen
FF17	Integer, dann umwandeln
******	Fließkommawert auf Null setzen
FF1B	
******	BASIC-Funktion SGN
FF2A	SGN
******	Akkuinhalt als Integerzahl übernehmen
FF32	Lo-Byte
FF33	Hi-Byte löschen
******	Integerzahl in HL übernehmen
FF35	Wert merken
FF38	Typ auf Integer
FF3A	und merken
******	Variablentyp auf Fließkomma
FF3E	Zeiger auf Fließkommazahl
FF41	Typ auf Real
******	Variablentyp holen, HL zeigt auf Variable
FF45	Zeiger auf Variable
FF48	Typ nach C
FF49	HL zeigt auf Variable
******	Variablentyp holen
FF4B	Variablentyp in Akku
******	Numerisches Ergebnis holen
FF4F	Variablentyp
FF52	String ?
FF54	ja, 'Type mismatch'

FF56	Intergerwert laden
FF59	kein Fließkomma, fertig
FF5A	Adresse der Fließkommazahl
FF5E	Test auf String
FF61	ja, ok
FF62	Fehlermeldung ausgeben
FF65	'Type mismatch'
******	Test auf String
FF66	Variablentyp
FF69	String ?
FF6C	Variablentyp setzen
FF6F	Adresse nach DE
******	Ergebnis auf BASIC-Stack ablegen
FF74	
FF76	Variablentyp
FF79	gleich Stackbedarf
FF7A	Platz im BASIC-Stack reservieren
FF7D	Ergebnis auf Stack ablegen
******	Variable nach (HL) kopieren
FF83	Zieladresse nach DE
FF84	Quelladresse
FF88	Variablentyp
FF8B	gleich Verschiebezähler
FF8C	Hi-Byte löschen
FF8E	verschieben
******	Test auf Buchstaben
FF92	Klein- in Großbuchstaben konvertieren
FF95	'A'
FF99	'Z'+1
******	Test auf alphanumerisches Zeichen
FF9C	Test auf Buchstabe
FF9F	ja
FFA0	'.'
FFA4	'0'
FFA8	'9'+1
******	Klein- in Großbuchstaben wandeln
FFAB	'a'
FFAE	'z'+1
FFB1	'a'-'A'

******	nachfolgende Tabelle durchsuchen
FFB4	
FFB6	Tabellenlänge laden
FFB8	Rücksprungadresse bei negativer Suche
FFBA	Zeiger auf nächstes Tabellenelement
FFBB	Zeichen vergleichen
FFBC	Zeiger erhöhen
FFBD	gefunden ?
FFBF	Tabelle noch nicht zu Ende ?
FFC1	Rücksprungadresse laden
FFC5	Adresse nach HL
******	Speicherbereich (HL) durchsuchen
FFCA	
FFCC	A nach C
FFCE	Null
FFD2	gleich ursprünglicher Akku
FFD4	Carry setzen
******	Vergleich HL <> DE
FFD8	
FFD9	H - D
FFDB	L - E
******	Vergleich HL <> BC
FFDE	
FFDF	H - B
FFE1	L - C
******	BC := HL - DE
FFE4	
FFE6	HL := HL - DE
FFE9	BC := HL
******	Blocktransfer LDIR, Anzahl in A
FFEC	Anzahl nach C
FFED	Hi-Byte auf Null
FFF0	Zähler BC = 0 ?
FFF1	ja, dann fertig
FFF2	Blocktransfer
******	Blocktransfer LDDR
FFF5	
FFF6	Zähler BC = 0 ?
FFF7	ja, dann fertig

FFF8	Blocktransfer
******	Sprung nach (HL)
FFFB	
******	Sprung nach (BC)
FFFC	
******	Sprung nach (DE)
FFFE	

4 Anhang

4.1 Die Betriebssystem-Routinen

Wir haben hier die Routinen und Tabellen des Betriebssystems ausgelistet, soweit sie uns bekannt sind.

Achtung: Versuchen Sie nie, die Routinen unter den hier erscheinenden Adressen anzuspringen, wenn Sie nicht mit dem Mechanismus zur Umschaltung der Speicherkonfiguration vertraut sind!

Benutzen Sie besser die im Kapitel 2.1 aufgeführten Vektoren.

Diese Aufstellung dient in erster Linie dazu, einen schnellen Überblick über das Betriebssystem zu bekommen. Aus diesem Grunde wurde hier nur die Aufstellung der Betriebssystem-Routinen des CPC 6128 (vgl. Kapitel 2.5) abgedruckt. Für den CPC 664 ergibt sich eine entsprechende Aufstellung mit geringfügigen Verschiebungen bei einigen Adressen.

KERNEL

0000	RST 0 RESET ENTRY
0008	RST 1 LOW JUMP
0010	RST 2 SIDE CALL
0018	RST 3 FAR CALL
0020	RST 4 RAM LAM
0028	RST 5 FIRM JUMP
0030	RST 6 USER RESTART
0038	RST 7 INTERRUPT ENTRY
0040	Bis hier wird ins RAM kopiert
0044	Restore High Kernel Jumps
005C	KL CHOKE OFF
0099	KL TIME PLEASE
00A3	KL TIME SET
00B1	Scan Events

0153	Kick Event
0163	KL NEW FRAME FLY
016A	KL ADD FRAME FLY
0170	KL DEL FRAME FLY
0176	KL NEW FAST TICKER
017D	KL ADD FAST TICKER
0183	KL DEL FAST TICKER
0189	Ticker Chain bearbeiten
01B3	KL ADD TICKER
01C5	KL DEL TICKER
01D2	KL INIT EVENT
01E2	KL EVENT
0219	KL DO SYNC
0227	KL SYNC RESET
022E	Sync Event einhängen
0255	KL NEXT SYNC
0276	KL DONE SYNC
0284	KL DEL SYNCHRONOUS
028D	KL DISARM EVENT
0294	KL EVENT DISABLE
029A	KL EVENT ENABLE
02A0	KL LOG EXT
02B1	KL FIND COMMAND
0326	KL ROM WALK
0330	KL INIT BACK
0379	Add Event
0388	Delete Event
0397	KL RAM-KONFIGURATION SETZEN
03C7	KL POLL SYNCHRONOUS
03E7	RST 7 INTERRUPT ENTRY CONT'D
041E	KL EXT INTERRUPT ENTRY
042A	KL LOW PCHL CONT'D
0430	RST 1 LOW JUMP CONT'D
045F	KL FAR PCHL CONT'D
0467	KL FAR ICALL CONT'D
046D	RST 3 LOW FAR CALL CONT'D
04BD	KL SIDE PCHL CONT'D
04C3	RST 2 LOW SIDE CALL CONT'D
04DB	RST 5 FIRM JUMP CONT'D

04F7	KL L ROM ENABLE CONT'D
04FE	KL L ROM DISABLE CONT'D
0505	KL U ROM ENABLE CONT'D
050C	KL U ROM DISABLE CONT'D
0516	KL ROM RESTORE CONT'D
051F	KL ROM SELECT CONT'D
0524	KL PROBE ROM CONT'D
052D	KL ROM DESELECT CONT'D
0543	KL CURR SELECTION CONT'D
0547	KL LDIR CONT'D
054D	KL LDDR CONT'D
0553	KL ROM OFF & KONFIG. SAVE
056C	RST 4 RAM LAM CONT'D
057D	KL RAM LAM (IX)

MASCHINE PACK

0591	Reset Cont'd
05C5	Tabelle 60Hz
05D5	Tabelle 50Hz
05ED	MC BOOT PROGRAM
061C	MC START PROGRAM
0677	Kaltstart
0688	Einschaltmeldung
06FC	Meldung ausgeben
0705	Ladefehler-Meldung
0738	Firmennamen
0776	MC SET MODE
0786	MC CLEAR INKS
078C	MC SET INKS
07AA	Farbe ausgeben
07B4	MC WAIT FLYBACK
07C0	MC SCREEN OFFSET
07E0	MC RESET PRINTER
07F7	Umlaute konvertieren
080C	MC ZEICHENZUORDNUNG
081B	MC PRINT CHAR
0835	MC WAIT PRINTER
0844	MC SEND PRINTER

0858	MC BUSY PRINTER
0863	MC SOUND REGISTER
0883	Scan Keyboard

JUMP RESTORE

08BD	JUMP RESTORE
08DE	Main Jump Adress
0A72	BASIC Jump Adr.
0AB4	Move (hl+3) nach ((hl+1)),cnt=(hl)

SCREEN PACK

0ABF	SCR INITIALISE
0AD0	SCR RESET
0AE9	SCR SET MODE
0B0C	SCR GET MODE
0B17	SCR CLEAR
0B37	SCR SET OFFSET
0B3C	SCR SET BASE
0B45	SCR VERÄNDERUNG SCREEN START
0B56	SCR GET LOCATION
0B5D	SCR CHAR LIMITS
0B6A	SCR CHAR POSTION
0BAF	SCR DOT POSITION
0C05	SCR NEXT BYTE
0C11	SCR PREV BYTE
0C1F	SCR NEXT LINE
0C39	SCR PREV LINE
0C55	SCR ACCESS
0C71	SCR WRITE
0C74	SCR PIXELS
0C7A	XOR Mode
0C7F	AND Mode
0C85	OR Mode
0C8A	SCR READ
0C8E	SCR INK ENCODE
0CA7	SCR INK DECODE
0CD8	Reset Farben

0CEA	SCR SET FLASHING
0CEE	SCR GET FLASHING
0CF2	SCR SET INK
0CF7	SCR SET BORDER
0CF8	Set Colour
0D10	Farbmatrix Eintrag holen
0D1A	SCR GET INK
0D1F	SCR GET BORDER
0D20	Get Colour
0D35	Ink Adresse holen
0D61	Set Inks on Frame Fly
0D73	Flash Inks
0D87	Params d. lfd Farbsatz holen
0D99	Farbmatrix
0DB9	SCR FILL BOX
0DBD	SCR FLOOD BOX
0DE5	SCR CHAR INVERT
0DF8	Farbspeicher adressieren
0E00	SCR HW ROLL
0E44	SCR SW ROLL
0EF9	SCR UNPACK
0F2A	SCR REPACK
0F93	SCR HORIZONTAL
0F9B	SCR VERTICAL
1052	Default Farben

TEXT SCREEN

1074	TXT INITIALISE
1084	TXT RESET
109F	Reset Params (alle Fenster)
10E4	TXT STR SELECT
1103	TXT SWAP STREAMS
111E	ldir cnt=15
1126	Adr. Fenster Params nach de
1139	Default Params setzen
115A	TXT SET COLUMN
1165	TXT SET ROW
1170	TXT SET CURSOR

117C	TXT GET CURSOR
1186	lfd. Fenster oben, links + hl
1193	lfd. Fenster oben, links - hl
11A4	Move Cursor
11CA	TXT VALIDATE
11D6	hl innerhalb Fenstergrenzen
1208	TXT WIN ENABLE
1252	TXT GET WINDOW
125F	TXT DRAW/UNDRAW CURSOR
1265	TXT PLACE/REMOVE CURSOR
1276	TXT CUR ON
127E	TXT CUR OFF
1286	TXT CUR ENABLE
1288	Cur Enable Cont'd
1297	TXT CUR DISABLE
1299	Cur Disable Cont'd
12A6	TXT SET PEN
12AB	TXT SET PAPER
12BA	TXT GET PEN
12C0	TXT GET PAPER
12C6	TXT INVERSE
12D4	TXT GET MATRIX
12F2	TXT SET MATRIX
12FE	TXT SET M TABLE
132B	TXT GET M TABLE
1335	TXT WR CHAR
134B	TXT WRITE CHAR
137B	TXT SET BACK
1388	TXT GET BACK
13A8	TXT SET GRAPHIC
13AC	TXT RD CHAR
13BE	TXT UNWRITE CHAR
13FE	TXT OUTPUT
140A	TXT OUT ACTION
1452	TXT VDU DISABLE
1459	TXT VDU ENABLE
1460	LFD. CURSOR FLAG NACH AKKU
1464	Default Steuerzeichen Sprünge kopieren
1474	Default Steuerzeichen Sprünge

14D4	TXT GET CONTROLS
14E1	Klingel
14EC	Transparentmode Ein/Aus
14F1	INK-Befehl
14FA	BORDER-Befehl
1501	Fenster definieren
150D	SYMBOL-Befehl
1519	CRSR Left
151E	CRSR Right
1523	CRSR Down
1528	CRSR Up
1539	CRSR Home
153F	CRSR auf Zeilenanfang
1547	LOCATE-Befehl
154F	TXT CLEAR WINDOW
155E	Zeichen auf CRSR-Pos. löschen
1565	Fenster ab CRSR-Pos. löschen
1578	Fenster bis CRSR-Pos. löschen
158F	Zeile ab CRSR-Pos. löschen
1599	Zeile bis CRSR-Pos. löschen

GRAPHICS SCREEN

15A8	GRA INITIALISE
15D7	GRA RESET
15EC	NN
15FB	GRA MOVE RELATIVE
15FE	GRA MOVE ABSOLUTE
1606	GRA ASK CURSOR
160E	GRA SET ORIGIN
161C	GRA GET ORIGIN
1624	phys Startposition holen
1627	phys Zielposition holen + Cur. setzen
162A	GRA KOORD. KONVERTIEREN
165D	Add lfd Koord. + rel Koord.
16A5	GRA WIN WIDTH
16EA	GRA WIN HEIGHT
1717	GRA GET W WIDTH
172D	GRA GET W HEIGHT

1736	GRA CLEAR WINDOW
1767	GRA SET PEN
176E	GRA SET PAPER
1775	GRA GET PEN
177A	GRA GET PAPER
1780	GRA PLOT RELATIVE
1783	GRA PLOT ABSOLUTE
1786	GRA PLOT
1794	GRA TEST RELATIVE
1797	GRA TEST ABSOLUTE
179A	GRA TEST
17A6	GRA LINE RELATIVE
17A9	GRA LINE ABSOLUTE
17AC	GRA MASK PARAM RETTEN
17B0	GRA MASK PARAM RETTEN
17B4	GRA LINE
1940	GRA WR CHAR
19D5	GRA PARAM RETTEN
19D9	GRA FILL

KEYBOARD MANAGER

1B5C	KM INITIALISE
1B98	KM RESET
1BBF	KM WAIT CHAR
1BC5	KM READ CHAR
1BFA	KM CHAR RETURN
1C04	KM EXP BUFFER
1C0A	Exp Buffer Cont'd
1C3C	Default Exp String
1C46	KM SET EXPAND
1C6A	Exp Buffer aufräumen
1CA7	Platz für neuen Exp String?
1CB3	KM GET EXPAND
1CC3	Adresse Exp String nach de
1CDB	KM WAIT KEY
1CE1	KM READ KEY
1D38	KM GET STATE
1D3C	Set State

1D40	KM UPDATE KEY STATE MAP
1DB8	KM TEST BREAK
1DE5	KM GET JOYSTICK
1DF2	KM GET DELAY
1DF6	KM SET DELAY
1DFA	KM ARM BREAK
1E0B	KM DISARM BREAK
1E19	KM BREAK EVENT
1E2F	KM GET REPEAT
1E34	KM SET REPEAT
1E45	KM TEST KEY
1E55	der Key# entspr. Bit holen
1E6D	Bit Masken
1EC4	KM GET TRANSLATE
1EC9	KM GET SHIFT
1ECE	KM GET CONTROL
1ED1	Get Key Table
1ED8	KM SET TRANSLATE
1EDD	KM SET SHIFT
1EE2	KM SET CONTROL
1EE5	Set Key Table
1EEF	Key Translation Table
1F3F	Key SHIFT Table
1F8F	Key CTRL Table

SOUND MANAGER

1FE9	SOUND RESET
2050	SOUND HOLD
206B	SOUND CONTINUE
208B	Sound Event
20D7	Scan Sound Queues
2114	SOUND QUEUE
21AC	SOUND RELEASE
21CE	SOUND CHECK
21EB	SOUND ARM EVENT
23DB	Lautstärke setzen
2495	SOUND AMPL ENVELOPE
249A	SOUND TONE ENVELOPE

249D	Hüllkurve kopieren
24A6	SOUND A ADRESS
24AB	SOUND T ADRESS
24AE	Hüllkurve Adresse holen

CASSETTE MANAGER

24BC	CAS INITIALISE
24CE	CAS SET SPEED
24E1	CAS NOISY
24E5	CAS IN OPEN
24FE	CAS OUT OPEN
2502	Cass. Open
2550	CAS IN CLOSE
2557	CAS IN ABANDON
257F	CAS OUT CLOSE
2599	CAS OUT ABANDON
25A0	CAS IN CHAR
25C6	CAS OUT CHAR
25F6	Check Input Buffer Status
25F9	Check Buffer Status
2603	CAS TEST EOF
2607	CAS RETURN
2618	CAS IN DIRECT
2653	CAS OUT DIRECT
2692	CAS CATALOG
26AC	File Header lesen
2891	Cass. Meldung (# in b) ausgeben
28F0	Cass. Meldung (1 Zeichen) ausgeben
2935	Kassetten-Meldungen
29A6	CAS READ
29AF	CAS WRITE
29C1	CAS CHECK
29E3	Motor Ein & Keyb. öffnen
2B3D	Cass. Input RD DATA & Test ESC
2BA7	Cass. Output WR DATA
2BBB	CAS START MOTOR
2BBF	CAS STOP MOTOR
2BC1	CAS RESTORE MOTOR

SCREEN EDITOR

2C02	EDIT
2C42	EDIT Sprung ausführen
2C72	EDIT Sprungtabelle 1
2CAE	EDIT Sprungtabelle 2
2CBD	CRSR UP
2CC1	CRSR DWN
2CC5	CRSR RGHT
2CC9	CRSR LEFT
2CD0	ESC
2CEA	BREAK-Meldung
2CF1	ENTER
2CFE	KLINGEL
2D02	CRSR RGHT (Puffer)
2D0A	CRSR DWN (Puffer)
2D14	CTRL & CRSR RGHT
2D1D	CTRL & CRSR DWN
2D34	CRSR LEFT (Puffer)
2D3C	CRSR UP (Puffer)
2D45	CTRL & CRSR LEFT
2D4F	CTRL & CRSR UP
2D81	CTRL & TAB (Filp Insert)
2D8A	Zeichen einfügen
2DC3	DEL
2DCD	CLR
2E17	SHFT & CRSR RGHT
2E1C	SHFT & CRSR LEFT
2E21	SHFT & CRSR UP
2E26	SHFT & CRSR DWN
2E65	COPY
2F56	Zeichen von Keyboard

ARITHMETIK

2F73	FLO PI
2F91	FLO VARIABLE VON (DE) NACH (HL) KOPIEREN
2F9F	FLO INTEGER NACH FLIESSKOMMA

2FC8	FLO 4-BYTE-WERT NACH FLIESSKOMMA
2FD1	FLO 4-BYTE-WERT MAL 256 NACH INTEGER
2FD9	FLO FLIESSKOMMA NACH INTEGER
3001	FLO FLIESSKOMMA NACH INTEGER
3014	FLO FIX
3055	FLO INT
305F	FLO
30C6	FLO ZAHL MIT 10^A MULTIPLIZIEREN
3136	FLO RND INIT
3143	FLO SET RANDOM SEED
3159	FLO RND
3188	FLO LETZTEN RND-WERT HOLEN
31B1	FLO LOG10
31B6	FLO LOG
322F	FLO EXP
32AC	FLO SQR
32AF	FLO POTENZIERUNG
3345	FLO DEG/RAD
3349	FLO COS
3353	FLO SIN
33C8	FLO TAN
33D8	FLO ATN
349E	FLO SUBTRAKTION
34A2	FLO ADDITION
3577	FLO MULITIPLIKATION
3604	FLO DIVISION
36DF	FLO VERGLEICH
3727	FLO SGN
3731	FLO VORZEICHENWECHSEL

CHARACTERS

3800-3FFF CHARACTERS

4.2 Referenzen zum System-RAM

Im Folgenden finden Sie zu jeder RAM-Adresse, soweit sie im Betriebssystem auftaucht, Querverweise auf die Stellen, an denen sie benutzt wird. Das ist dann sehr hilfreich, wenn Sie die Inhalte der RAM-Adressen mit eigenen Programmen manipulieren und plötzlich ein anderer Wert als erwartet darinsteht.

Auch hier beschränken wir uns wieder auf die Gegebenheiten beim CPC 6128.

```
B100:  0638
B101:  063B
B114:  2DA5 2DBB 2DDE 2DEA
B115:  2C24 2D81 2D85 2D8D
B116:  2DF3 2DFA 2E13 2E41 2EC1
B117:  2DF6
B118:  24E1 2807 28D2
B119:  280C 290F
B11A:  24E5 2550 2557 25F6 2692 26E0 271B 292F
B11B:  263C 269C 26EF
B11D:  25BC 25C1 260F 2613 26F2
B11F:  2743 274E 2760
B12F:  26FC
B130:  26AC
B131:  24FA
B132:  25AA 25B5 25B9 2608 260C 2629 263F 270C
B134:  24F2 261F 2626 26DD
B136:  2706
B137:  24F6
B15F:  24FE 257F 2599 25CA 2656 27D9
B160:  266E 2685 279E
B162:  25EA 25EF 27A1
B164:  2790 27A8
B174:  27CD
B175:  258B 27BF
```

B176: 2663
B177: 25D4 25E3 25E7 2671 267E 27B6 27CA
B179: 27A4
B17B: 2796 27D2
B17C: 2666
B17E: 266A
B1A4: 26BB 274B 2763 2804
B1B5: 2700
B1B7: 26D9 2709
B1B9: 2022 2072 2094 20BE 2122 214D 21B9
B1BB: 273D
B1BC: 21D1
B1BE: 20E9 2637
B1D5: 21EF
B1E4: 2564 27E5
B1E5: 29E3 2ACD 2AE3
B1E6: 2AC6 2B23
B1E7: 24DC
B1E8: 2B78 2B8B
B1E9: 24D9
B1EA: 2B7C
B1EB: 2B00 2B12 2B16
B1ED: 1FE9 206B
B1EE: 2050 208D 20B7 20D7 2258 2286
B1F0: 201D 20D1 210C 2147 21B4
B1F8: 2000 2296
B237: 229E 22C0
B276: 22A6 22B8
B2A6: 2303 2495 24A6
B2B5: 1FFD 23EF
B396: 249A 24AB
B590: 1B9E
B5D6: 1B6E
B628: 1BCF 1BF0
B629: 1C38
B62A: 1BC6 1BFA
B62B: 1C17 1CC9
B62D: 1C13
B62F: 1C35 1C96 1CA1 1CA7

B630: 1CAC
B631: 1B68 1D12 1D2B 1D38 1D3C
B632: 1CFB
B633: 1D9E 1DF2 1DF6
B634: 1DD8
B635: 1B8A 1D57 1D86 1E4D
B637: 1D4F 1E46
B63B: 1DE5
B63D: 1DB8
B63E: 1DEB
B63F: 1B8D 1D43
B649: 1D40 1D54
B64B: 1D49
B653: 1D7A 1D92 1DA1
B654: 1D7F 1DAC
B655: 1B63
B656: 1E0D 1E19
B657: 1DFD
B686: 1E76 1E86 1EAE
B688: 1E97 1E9D
B68A: 1D96 1E93 1EAA
B68B: 1EC4 1ED8
B68D: 1EC9 1EDD
B68F: 1ECE 1EE2
B691: 1D8B 1E2F 1E37
B692: 1B71
B693: 160E 161C 1640
B695: 1612 1620 1655
B697: 15FE 1606 165E
B699: 1602 160A 1664
B69B: 166A 16C9 1717 1753 1910
B69D: 1673 16CD 171B 1906
B69F: 1680 16FB 172D 174A 18B9 1AE8
B6A1: 1689 16FF 1731 1746 18C3 1B18
B6A3: 0FA5 0FAE 0FB1 0FFF 101C 176A 1775 178D 19C4 1B34
B6A4: 0FF3 1027 175D 1771 177A 19CE
B6A5: 17BD 188F 18C8 18DA 18E6 18EF 18FA 18FF 19D9 1A19 1A44 1AAC 1AC1
B6A7: 17CC 1893 18A2 18AD 18B2 1915 1928 1934 19DF 1A25 1A2C 1A9F 1AA9
B6A9: 1802 1861 19FE 1A4B 1AC6

B6AA: 19E6 1B3A
B6AB: 17EC 1846 1A0B 1A21 1ABD 1AD7 1ADB 1ADF
B6AC: 1A50 1A79
B6AD: 17C4 17E8 1812 181B 18D2 18DD
B6AE: 17D3 17E2 191F 1A5D 1A66 1A94
B6AF: 17DF 1828 1898
B6B0: 17F9 1868 1876 1880 1A76 1A97
B6B2: 17B0 17F2 1820
B6B3: 0FA9 0FB4 0FBA 1012 104C 17AC
B6B4: 0FF7 1021 19C9 19D5
B6B5: 10AF 10B3 10E6 1103 110C
B6B6: 10A1
B726: 10A4 1135 115F 116A 1176 117C 11A7 11AD 1340 1555 156F 1582
B728: 123A 1259
B729: 1166 1186 1193 11EF 1229 1252 1539 1552 1568 157B
B72A: 115B 118C 119B 11DD 11E2 1542 159E
B72B: 11F7 122C 1255 1558 156B
B72C: 11D6 11EA 157E 1593
B72D: 1182 11B2
B72E: 113C 125F 128E 129F 1336 143B 1460
B72F: 10CA 10DA 126B 12A6 12BA 12C9 12CF 1392 13A0 13DB
B730: 11BD 12AB 12C0 13BE 1589
B731: 1377 1384 1388
B733: 13A8 140B
B734: 1321 132B
B735: 1078
B736: 1326 1331
B738: 134F 13C1 13E7
B758: 1413 144E 1465
B759: 142C 1446
B763: 146B
B7C3: 0B0C 0B31
B7C4: 0B3C 0B51 0B56 0B8A 0E2A 0E3D
B7C5: 0B20
B7C6: 0AC7 0B37 0B47 0B59 0B93 0BED 0E32
B7C7: 0C6A 0C71
B7C8: 0C6D
B7D2: 0CEA 0CEE 0D95
B7D3: 0D8E

```
B7D4:  0CDB 0D92
B7E5:  0D38 0D87
B7F6:  0CE4 0D7C 0D8A
B7F7:  0D0C 0D83
B7F8:  0D61 0D73
B7F9:  0D42 0D55
B802:  0FA1 0FBD
B804:  07E3 0812
B82D:  0066 00F2 011D 0127
B82E:  00EC
B82F:  00F5 00FE 0102
B831:  00E2 00F8 0114 0132 0142 03FE
B832:  010A 014E
B8B4:  009E 00AC 00B1 010E
B8B6:  009A 00A8
B8B8:  00A5
B8B9:  00BF 016A 0170
B8BB:  00C7 017D 0183
B8BD:  00DC 0189 01BF 01C5
B8BF:  00D2 03D0
B8C0:  0256 026E 0287 03D6
B8C1:  022A 03C7
B8C2:  0263 026B 0276 0294 029A 03E0
B8C3:  0230 02B1 0307
B8D3:  02A1 02A5 02BE
B8D5:  0399
B8D6:  0080 0351 0484 04B5 0539 0543
B8D7:  0060 0086
B8D9:  005D 0083 0330 04D5
B8DA:  034E
```

00	Zeilenende	93	DIM
01	':', Ende des Statements	94	DRAW
02	Integervariable '%'	95	DRAWR
03	Stringvariable '$'	96	EDIT
04	Realvariable '!'	97	ELSE
0D	Variable ohne Kennzeichen	98	END
0E	Konstante 0	99	ENT
0F	Konstante 1	9A	ENV
10	Konstante 2	9B	ERASE
11	Konstante 3	9C	ERROR
12	Konstante 4	9D	EVERY
13	Konstante 5	9E	FOR
14	Konstante 6	9F	GOSUB
15	Konstante 7	A0	GOTO
16	Konstante 8	A1	IF
17	Konstante 9	A2	INK
19	Ein-Byte-Wert	A3	INPUT
1A	Zwei-Byte-Wert, dezimal	A4	KEY
1B	Zwei-Byte-Wert, binär	A5	LET
1C	Zwei-Byte-Wert, hex	A6	LINE
1D	Zeilenadresse	A7	LIST
1E	Zeilennummer	A8	LOAD
1F	Fließkommawert	A9	LOCATE
80	AFTER	AA	MEMORY
81	AUTO	AB	MERGE
82	BORDER	AC	MID$
83	CALL	AD	MODE
84	CAT	AE	MOVE
85	CHAIN	AF	MOVER
86	CLEAR	B0	NEXT
87	CLG	B1	NEW
88	CLOSEIN	B2	ON
89	CLOSEOUT	B3	ON BREAK
8A	CLS	B4	ON ERROR GOTO 0
8B	CONT	B5	ON SQ
8C	DATA	B6	OPENIN
8D	DEF	B7	OPENOUT
8E	DEFINT	B8	ORIGIN
8F	DEFREAL	B9	OUT
90	DEFSTR	BA	PAPER
91	DEG	BB	PEN
92	DELETE	BC	PLOT

BD	PLOTR
BE	POKE
BF	PRINT
C0	'
C1	RAD
C2	RANDOMIZE
C3	READ
C4	RELEASE
C5	REM
C6	RENUM
C7	RESTORE
C8	RESUME
C9	RETURN
CA	RUN
CB	SAVE
CC	SOUND
CD	SPEED
CE	STOP
CF	SYMBOL
D0	TAG
D1	TAGOFF
D2	TRON
D3	TROFF
D4	WAIT
D5	WEND
D6	WHILE
D7	WIDTH
D8	WINDOW
D9	ZONE
DA	WRITE
DB	DI
DC	EI
DD	FILL
DE	GRAPHICS
DF	MASK
E0	FRAME
E1	CURSOR
E3	ERL
E4	FN
E5	SPC
E6	STEP
E7	SWAP
EA	TAB
EB	THEN
EC	TO
ED	USING
EE	>
EF	=
F0	>=
F1	<
F2	<>
F3	<=
F4	+
F5	-
F6	*
F7	/
F8	^
F9	'Backslash'
FA	AND
FB	MOD
FC	OR
FD	XOR
FE	NOT
FF	Funktion

Das Token &FF steht vor einer Funktion. Danach können die nachstehenden Token folgen:

00	ABS	71	BIN$
01	ASC	72	DEC$
02	ATN	73	HEX$
03	CHR$	74	INSTR
04	CINT	75	LEFT$
05	COS	76	MAX
06	CREAL	77	MIN
07	EXP	78	POS
08	FIX	79	RIGHT$
09	FRE	7A	ROUND
0A	INKEY	7B	STRING$
0B	INP	7C	TEST
0C	INT	7D	TESTR
0D	JOY	7E	COPYCHR$
0E	LEN	7F	VPOS
0F	LOG		
10	LOG10		
11	LOWER$		
12	PEEK		
13	REMAIN		
14	SGN		
15	SIN		
16	SPACE$		
17	SQ		
18	SQR		
19	STR$		
1A	TAN		
1B	UNT		
1C	UPPER$		
1D	VAL		
40	EOF		
41	ERR		
42	HIMEM		
43	INKEY$		
44	PI		
45	RND		
46	TIME		
47	XPOS		
48	YPOS		
49	DERR		

MONITOR

Wir können uns gut vorstellen, daß es dem einen oder anderen unter Ihnen unter den Fingern juckt, zu erfahren, was denn nun im einzelnen hinter dem Rom-Listing steckt, welches ja in Wirklichkeit nur den symbolischen Inhalt des Betriebssystems wiedergibt. Aber leider haben die Götter vor den Erfolg den Schweiß gesetzt, und so wird Ihnen, sofern Sie nicht bereits über einen komfortableren Monitor verfügen, nichts anderes übrig bleiben, als den hier veröffentlichten abzutippen.
Bis auf zwei kleine Maschinenroutinen, die einmal zum Lesen eines Bytes aus dem Speicher, zum anderen zum Holen eines Bytes aus einer Datei dienen, ist das Programm komplett in Basic geschrieben. Dadurch, daß der gesamte Befehlsvorrat zunächst in Arrays eingelesen wird, ist der Disassembler dennoch ganz schön flott.
Ein Mangel sei nicht verschwiegen: Das gewählte Verfahren ist nicht in der Lage, bestimmte Befehle des Typs (IX+xx) zu verarbeiten. Taucht ein solcher auf, erscheint im Listing die Meldung "!! Spezialbefehl ....". Bei Bedarf müssen Sie sich dann anhand des Bitmusters den Befehl selbst zusammenreimen. Solche Befehle sind allerdings recht selten. Sie tauchen nur im Sound-Manager zwei- oder dreimal auf.
Außerdem entspricht die Darstellung der Befehle nicht ganz dem Z80-Standard. So werden bei uns z.B. Immediate-Werte durch ein vorangestelltes Doppelkreuz gekennzeichnet. Doppelbytewerte ohne ein solches sind Adressen.
Sie haben die Möglichkeit, aus Ram, Rom oder Datei zu disassemblieren. Die letztere Möglichkeit dürfte Ihnen so leicht kein anderes Programm bieten und ist dann sinnvoll anzuwenden, wenn das zu bearbeitende Programm sich nicht zusammen mit einem Basic-Programm im Speicher verträgt.
Bevor wir mit der Befehlsbeschreibung beginnen, noch ein kleiner Tip: Lassen Sie zunächst die Zeilen 20-40 weg, damit beim Probelauf des Programmes ein durch Tippfehler verursachter Syntax-Error nicht unterdrückt wird. Falls Sie ohnehin nicht vorhaben, von einer Datei zu arbeiten, können diese Zeilen auch wegbleiben, denn sie dienen nur dazu, das

beim Öffnen einer Datei sonst unvermeidliche 'Aufräumen' des Speichers zu verhindern. Hieraus entnehmen Sie bitte auch, daß Sie das Programm "mimo.bas" nennen müssen, damit der OPENIN auch eine Datei findet.
Nun zu den wenigen Befehlen. Grundsätzlich gilt, daß etwaige Parameter unmittelbar hinter dem Kommando in HEX eingegeben werden werden. Wollen Sie z.B. die laufende Adresse auf $0048 setzen, so geben Sie ein: m48>ENTER<

d Disassemblieren ab der laufenden Adresse. Diese Funktion wird durch Drücken irgendeiner Taste abgebrochen.

f Hinter dem f schließt sich sofort der vollständige Dateiname an, den Sie behandeln wollen. Mit der folgenden Eingabe setzen Sie die relative Adresse, mit der die Datei auf dem Bildschirm erscheinen soll. Dies dient nur der Optik. Die Datei selbst wird in jedem Falle von vorne begonnen. Nachfolgende Anzeigebefehle beziehen sich dann auf diese Datei. Der Dateimodus wird durch die Funktion m abgebrochen.

i Bytes in den Speicher schreiben. Dieser Befehl verlangt keine weiteren Parameter. Die Bytes werden ab der laufenden Adresse einzeln abgefordert. Diese Funktion wird mit einer Leereingabe abgebrochen.

o Ausgabefile einstellen. 0 ist der Normalfall und bringt alle Anzeigen im Mode 1 auf den Bildschirm. 1 bringt den Bildschirm in Mode 2 und teilt ihn, so daß das obere Drittel für die einfache Speicheranzeige zuständig ist, der Rest für den Disassembler. Beim Wechsel von Anzeige auf Disassembler und umgekehrt bleiben die Fenster erhalten. 8 schließlich lenkt die Ausgabe auf den Drucker.

m stellt die laufende Adresse ein, auf die sich alle nachfolgenden Befehle beziehen.

b stellt die Speicherkonfiguration ein. Das verlangte Byte hat den Aufbau, wie an anderer Stelle in diesem Buch beschrieben. FE z.B. selektiert die beiden eingebauten Roms und den dazwischen liegenden Ram, FF wählt nur den Ram aus.

$ wandelt den dezimalen Parameter in Hex um.

% wandelt den Hex-Parameter (max. vierstellig) in eine Dezimalzahl.

x beendet das Programm und setzt die Speichergrenze zurück.

? macht einen Warmstart und zeigt die Befehlsübersicht an

>ENTER< alleine eingegeben listet den Speicherinhalt in Hex und ASCII.

Bleibt uns nur noch zu hoffen, daß Ihnen das Abtippen des folgenden Listing nicht allzuviel Mühe bereitet. Ein '^' im Listing entspricht übrigens dem 'Pfeil nach oben'.

```
10 top=HIMEM
20 ON ERROR GOTO 40
30 OPENIN "mimo.bas"
40 RESUME NEXT
50 MEMORY HIMEM-1
60 CLOSEIN
70 him=HIMEM-256
80 ZONE 8:lf=0
90 mpb=him-20:MEMORY mpb-1
100 GOSUB 1350:ms=&FE
110 CLS:INK 3,6:b0=1:b1=24:b2=22:b3=0
120 DIM l%(4,255),mn$(4,255),pu%(15)
130 GOSUB 1010:a=0
140 bs$=STRING$(32,8):bl$=SPACE$(30)
150 IF plf=1 THEN lf=0:plf=0
160 MODE 1:PRINT:PRINT: PRINT"c = Call Maschinenprogramm"
170 PRINT"d = Diassemblieren"
180 PRINT"f = File"
190 PRINT"i = Insert Bytes"
200 PRINT"o = Output-lf#"
210 PRINT"m = Memoryadress"
220 PRINT"b = Bank-select
230 PRINT"$ = Dezimal -> Hex"
240 PRINT"% = Hex -> Dezimal"
250 PRINT"x = Ende"
260 PRINT"? = Warmstart"
270 PRINT:GOTO 290
280 IF lf=0 OR lf>7 THEN MODE 1
290 BORDER b0:INK 0,b0:INK 1,b1:PRINT:PRINT"bank= ";HEX$(ms,2):PRINT "mem = ";
    HEX$(a,4):i=a:PRINT"lf# =";lf:PRINT
300 INPUT">",h$:hl$=LEFT$(h$,1)
310 IF h$="?" THEN GOTO 150
320 IF h$="x" THEN MEMORY top:MODE 1:END
330 IF hl$<>"o" THEN 370
340 lf=VAL(RIGHT$(h$,1)):IF lf=0 OR lf>7 THEN plf=0:GOTO 280
350 IF plf=0 THEN MODE 2:WINDOW #0,1,80,25,25:WINDOW #1,1,80,1,8:
    WINDOW #2,1,80,9,25:plf=1
360 GOTO 290
```

```
370 IF hl$="$" THEN PRINT HEX$(VAL(RIGHT$(h$,LEN(h$)-1))):GOTO 290
380 IF hl$<>"%" THEN 410
390 xx=(VAL("&"+RIGHT$(h$,LEN(h$)-1))):IF xx<0 THEN xx=xx+65536
400 PRINT xx:GOTO 290
410 IF hl$<>"m" THEN 460
420 IF file=1 THEN file=0:CLOSEIN
430 IF LEN(h$)=1 THEN 280
440 a=VAL("&"+RIGHT$(h$,LEN(h$)-1)):IF a<0 THEN a=a+65536
450 padp=a-1:GOTO 280
460 IF hl$<>"b" THEN 490
470 re=VAL("&"+RIGHT$(h$,LEN(h$)-1)):IF re>255 OR re<0 THEN
    PRINT"2-Byte Hexwert verlangt":GOTO 280
480 ms=re:GOTO 280
490 IF hl$<>"f" THEN 570
500 IF file=1 THEN CLOSEIN
510 ON ERROR GOTO 530
520 OPENIN MID$(h$,2)
530 RESUME NEXT
540 INPUT"basis (hex) ";h$
550 h$="m"+h$
560 file=1:GOTO 440
570 REM
580 IF hl$="d" THEN i=a:GOTO 810
590 IF hl$="c" THEN CALL a:GOTO 280
600 IF hl$="i" THEN 780
610 IF LEN(h$)<2 THEN h$="00"
620 bis=VAL("&"+RIGHT$(h$,LEN(h$)-1)):IF bis<1 THEN bis=bis+65536
630 IF plf=0 THEN MODE 2 ELSE lf=1
640 BORDER b2:INK 0,b2:INK 1,b3
650 ON file GOTO 670
660 a=INT(a/16)*16
670 FOR i=a TO bis  STEP 16
680 PRINT#lf,HEX$(i,4);":";:FOR j=0 TO 15
690 pad=i+j:GOSUB 1520:PRINT#lf," ";HEX$(mv,2);
700 NEXT j:PRINT#lf,TAB(60);
710 FOR j=0 TO 15:pad=i+j:GOSUB 1520:he=(mv AND 127)
720 IF he<32 OR he=127 THEN he=46
730 PRINT#lf,CHR$(he);:NEXT j:PRINT#lf
740 IF INKEY$<>"" THEN a=i:i=65535:ELSE a=i+16
```

```
750 NEXT
760 IF lf<>8 THEN INPUT ">ENTER< druecken, wenn fertig";re$
770 GOTO 280
780 i=a
790 PRINT HEX$(i,4);": ";:INPUT"",d$:IF d$="" THEN 280
800 POKE i,VAL("&"+d$):i=i+1:GOTO 790
810 IF plf=1 THEN lf=2:PRINT#lf,CHR$(11);
820 IF LEN(h$)=1 THEN h$="00"
830 bis=VAL("&"+RIGHT$(h$,LEN(h$)-1)):IF bis<1 THEN bis=bis+65536
840 pa=a
850 PAPER 0:IF INKEY$ <>"" THEN a=pa:PRINT#lf:GOTO 280
860 IF pa>bis THEN a=pa:PRINT#lf:GOTO 760
870 pad=pa:GOSUB 1520:op=mv:ad=pa:pa=pa+1
880 IF lf=8 THEN PRINT#lf,LEFT$(bl$,10);
890 PRINT#lf,HEX$(ad,4);"  ";:xx=0
900 PRINT#lf,HEX$(op,2);
910 se=0:GOSUB 1700:IF LEFT$(mn$,1)="?" THEN 1070
920 se=xx:GOSUB 1700:IF mn$="" THEN PAPER 3:PRINT#lf,"        ????":
    PAPER 0:GOTO 850
930 ON l%(xx,op) GOTO 980,970,960,950
940 ON l%(xx,op)-1 GOTO 980,970,960,950
950 pad=pa:GOSUB 1520:PRINT#lf,HEX$(mv,2);:pa=pa+1
960 pad=pa:GOSUB 1520:PRINT#lf,HEX$(mv,2);:pa=pa+1
970 pad=pa:GOSUB 1520:PRINT#lf,HEX$(mv,2);:pa=pa+1
980 PRINT#lf,LEFT$(bl$,(4-l%(xx,op))*2+2);
990 GOSUB 1090
1000 GOTO 850
1010 PRINT:PAPER 3:PRINT"bitte warten";:PAPER 0:PRINT:FOR i=0 TO 4:
     FOR j=0 TO 255
1020 READ a:l%(i,j)=a
1030 NEXT j,i
1040 FOR i=0 TO 4:FOR j=0 TO 255
1050 READ mn$:mn$(i,j)=mn$
1060 NEXT:NEXT:RETURN
1070 xx=l%(0,op):pad=pa:GOSUB 1520:op=mv:se=xx:GOSUB 1700:IF mn$="" THEN 920
1080 PRINT#lf,HEX$(op,2);:pa=pa+1:GOTO 940
1090 se=xx:GOSUB 1700:ln=LEN(mn$)
1100 IF mn$=pmn$ THEN PAPER 3
1110 pmn$=mn$:ppn=1
```

```
1120 IF MID$(mn$,ln-3,4)="+/-^" THEN mn$=LEFT$(mn$,ln-4):GOTO 1230
1130 pn=INSTR(mn$,"*"):IF pn<>0 THEN PRINT#lf,LEFT$(mn$,pn-1);:GOTO 1170
1140 pn=INSTR(ppn,mn$,"^"):IF pn<>0 THEN PRINT#lf,MID$(mn$,ppn,pn-ppn);:
     GOTO 1220
1150 PRINT#lf,mn$;
1160 PRINT#lf:RETURN
1170 pad=pa-2:GOSUB 1520:ar=mv:pn=pn+1
1180 IF pn>ln THEN xz=ar:PRINT#lf,HEX$(xz,2);:GOTO 1160
1190 ppn=pn:IF MID$(mn$,pn,1)<>"^" THEN xz=ar:PRINT#lf,HEX$(xz,2);:GOTO 1140
1200 pn=pn+1:pad=pa-1:GOSUB 1520:yy=256*mv+ar:PRINT#lf,HEX$(yy,4);
1210 PRINT#lf,MID$(mn$,pn):RETURN
1220 pn=pn+1:pad=pa-1:GOSUB 1520:ar=mv:xz=ar:PRINT#lf,HEX$(xz,2);:GOTO 1210
1230 PRINT#lf,mn$;
1240 pn=pn+1:pad=pa-1:GOSUB 1520:ar=mv:yy=ad+2+ar+(ar>127)*256:
     PRINT#lf,HEX$(yy,4);
1250 PRINT#lf:RETURN
1260 sp=1
1270 WHILE MID$(mn$,sp,1)<>" ": sp=sp+1:WEND
1280 WHILE MID$(mn$,sp,1)=" ": sp=sp+1:WEND
1290 ad=cn+VAL(RIGHT$(mn$,LEN(mn$)-sp+1))
1300 ha=INT(ad/256):la=ad-ha*256
1310 PRINT#lf,"  ($";HEX$(ha,2);HEX$(la,2);")":RETURN
1320 IF MID$(mn$,sp,1)="-" THEN 1340
1330 ad=cn+ar:GOTO 1300
1340 ad=cn+ar-256:GOTO 1300
1350 POKE mpb,&DF
1360 po=mpb+4:ph=INT(po/256):pl=po-ph*256
1370 POKE mpb+1,pl:POKE mpb+2,ph
1380 POKE mpb+3,&C9
1390 po=mpb+7:ph=INT(po/256):pl=po-ph*256
1400 POKE mpb+4,pl:POKE mpb+5,ph
1410 POKE mpb+7,&3A
1420 by=mpb+14:ph=INT(by/256):pl=by-ph*256
1430 POKE mpb+10,&32
1440 POKE mpb+11,pl:POKE mpb+12,ph
1450 POKE mpb+13,&C9
1460 DATA &c1,&d1,&f1,&e1,&f5,&d5,&c5,&cd,&80,&bc,&f5,&d1,&72,&23,&73,&c9
1470 FOR i=1 TO 16
1480 READ a
```

```
1490 mp$=mp$+CHR$(a)
1500 NEXT i
1510 RETURN
1520 IF pad>65535 THEN RETURN
1530 ON file GOTO 1600
1540 IF ms=255 THEN mv=PEEK(pad):RETURN
1550 ph=INT(pad/256):pl=pad-ph*256
1560 POKE mpb+8,pl:POKE mpb+9,ph
1570 POKE mpb+6,ms
1580 CALL mpb
1590 mv=PEEK(by):RETURN
1600 IF padp<pad THEN GOSUB 1630
1610 mv=pu%(pad MOD(16))
1620 RETURN
1630 ret%=0:mpp=@mp$
1640 getf=PEEK(mpp+1)+256*PEEK(mpp+2)
1650 CALL getf,@ret%
1660 mv=ret% AND 255: IF (ret% AND &100)=0 THEN mv=0
1670 padp=padp+1:pu%(padp MOD(16))=mv
1680 IF padp<pad GOTO 1650
1690 RETURN
1700 mn$=mn$(se,op):RETURN
1710 DATA 1 , 3 , 1 , 1 , 1 , 1 , 2 , 1
1720 DATA 1 , 1 , 1 , 1 , 1 , 1 , 2 , 1
1730 DATA 2 , 3 , 1 , 1 , 1 , 1 , 2 , 1
1740 DATA 2 , 1 , 1 , 1 , 1 , 1 , 2 , 1
1750 DATA 2 , 3 , 3 , 1 , 1 , 1 , 2 , 1
1760 DATA 2 , 1 , 3 , 1 , 1 , 1 , 2 , 1
1770 DATA 2 , 3 , 3 , 1 , 1 , 1 , 2 , 1
1780 DATA 2 , 1 , 3 , 1 , 1 , 1 , 2 , 1
1790 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1800 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1810 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1820 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1830 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1840 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1850 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1860 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1870 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
```

```
1880 DATA 0 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1890 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1900 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1910 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1920 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1930 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1940 DATA 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1
1950 DATA 1 , 1 , 3 , 3 , 3 , 1 , 2 , 1
1960 DATA 1 , 1 , 3 , 4 , 3 , 3 , 2 , 1
1970 DATA 1 , 1 , 3 , 2 , 3 , 1 , 2 , 1
1980 DATA 1 , 1 , 3 , 2 , 3 , 2 , 2 , 1
1990 DATA 1 , 1 , 3 , 1 , 3 , 1 , 2 , 1
2000 DATA 1 , 1 , 3 , 1 , 3 , 1 , 2 , 1
2010 DATA 1 , 1 , 3 , 1 , 3 , 1 , 2 , 1
2020 DATA 1 , 1 , 3 , 1 , 3 , 3 , 2 , 1
2030 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2040 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2050 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2060 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2070 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2080 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2090 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2100 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2110 DATA 2 , 2 , 2 , 4 , 2 , 2 , 2 , 2
2120 DATA 2 , 2 , 2 , 4 , 0 , 2 , 0 , 2
2130 DATA 2 , 2 , 2 , 4 , 0 , 0 , 2 , 2
2140 DATA 2 , 2 , 2 , 4 , 0 , 0 , 2 , 2
2150 DATA 2 , 2 , 2 , 4 , 0 , 0 , 0 , 2
2160 DATA 2 , 2 , 2 , 4 , 0 , 0 , 0 , 2
2170 DATA 2 , 0 , 2 , 4 , 0 , 0 , 0 , 0
2180 DATA 2 , 2 , 2 , 4 , 0 , 0 , 0 , 0
2190 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2200 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2210 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2220 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2230 DATA 2 , 2 , 2 , 2 , 0 , 0 , 0 , 0
2240 DATA 2 , 2 , 2 , 2 , 0 , 0 , 0 , 0
2250 DATA 2 , 2 , 2 , 2 , 0 , 0 , 0 , 0
2260 DATA 2 , 2 , 2 , 2 , 0 , 0 , 0 , 0
```

```
2270 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2280 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2290 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2300 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2310 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2320 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2330 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2340 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2350 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2360 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2370 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2380 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2390 DATA 0 , 4 , 4 , 2 , 0 , 0 , 0 , 0
2400 DATA 0 , 2 , 4 , 2 , 0 , 0 , 0 , 0
2410 DATA 0 , 0 , 0 , 0 , 3 , 3 , 4 , 0
2420 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2430 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2440 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2450 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2460 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2470 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2480 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2490 DATA 3 , 3 , 3 , 3 , 3 , 3 , 0 , 3
2500 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2510 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2520 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2530 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2540 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2550 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2560 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2570 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2580 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2590 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2600 DATA 0 , 0 , 0 , 4 , 0 , 0 , 0 , 0
2610 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2620 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2630 DATA 0 , 2 , 0 , 2 , 0 , 2 , 0 , 0
2640 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2650 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
```

```
2660 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2670 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2680 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2690 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2700 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2710 DATA 0 , 4 , 4 , 2 , 0 , 0 , 0 , 0
2720 DATA 0 , 2 , 4 , 2 , 0 , 0 , 0 , 0
2730 DATA 0 , 0 , 0 , 0 , 3 , 3 , 4 , 0
2740 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2750 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2760 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2770 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2780 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2790 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2800 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2810 DATA 3 , 3 , 3 , 3 , 3 , 3 , 0 , 3
2820 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2830 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2840 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2850 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2860 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2870 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2880 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2890 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2900 DATA 0 , 0 , 0 , 0 , 0 , 0 , 3 , 0
2910 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2920 DATA 0 , 0 , 0 , 4 , 0 , 0 , 0 , 0
2930 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2940 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2950 DATA 0 , 2 , 0 , 2 , 0 , 2 , 0 , 0
2960 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2970 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
2980 DATA 0 , 2 , 0 , 0 , 0 , 0 , 0 , 0
2990 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3000 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3010 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3020 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3030 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3040 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
```

```
3050 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
3060 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3070 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3080 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3090 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3100 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3110 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3120 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3130 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3140 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3150 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3160 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3170 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3180 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3190 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3200 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3210 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3220 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3230 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3240 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3250 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3260 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3270 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3280 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3290 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3300 DATA 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2
3310 DATA "nop      ","ld      bc,#*^","ld      (bc),a","inc     bc",
     "inc     b","dec     b","ld      b,#^","rlca    "
3320 DATA "ex      af,af'","add     hl,bc","ld      a,(bc)","dec     bc",
     "inc     c","dec     c","ld      c,#^","rrca    "
3330 DATA "djnz    +/-^","ld      de,#*^","ld      (de),a","inc     de",
     "inc     d","dec     d","ld      d,#^","rla     "
3340 DATA "jr      +/-^","add     hl,de","ld      a,(de)","dec     de",
     "inc     e","dec     e","ld      e,#^","rra     "
3350 DATA "jr      nz,+/-^","ld      hl,#*^","ld      *^,hl","inc     hl",
     "inc     h","dec     h","ld      h,#^","daa     "
3360 DATA "jr      z,+/-^","add     hl,hl","ld      hl,*^","dec     hl",
     "inc     l","dec     l","ld      l,#^","cpl     a"
3370 DATA "jr      nc,+/-^","ld      sp,#*^","ld      *^,a","inc     sp",
```

```
     "inc     (hl)","dec     (hl)","ld      (hl),#^","scf     "
3380 DATA "jr      c,+/-^","add     hl,sp","ld      a,*^","dec     sp",
     "inc     a","dec     a","ld      a,#^","ccf     "
3390 DATA "ld      b,b","ld      b,c","ld      b,d","ld      b,e",
     "ld      b,h","ld      b,l","ld      b,(hl)","ld      b,a"
3400 DATA "ld      c,b","ld      c,c","ld      c,d","ld      c,e",
     "ld      c,h","ld      c,l","ld      c,(hl)","ld      c,a"
3410 DATA "ld      d,b","ld      d,c","ld      d,d","ld      d,e",
     "ld      d,h","ld      d,l","ld      d,(hl)","ld      d,a"
3420 DATA "ld      e,b","ld      e,c","ld      e,d","ld      e,e",
     "ld      e,h","ld      e,l","ld      e,(hl)","ld      e,a"
3430 DATA "ld      h,b","ld      h,c","ld      h,d","ld      h,e",
     "ld      h,h","ld      h,l","ld      h,(hl)","ld      h,a"
3440 DATA "ld      l,b","ld      l,c","ld      l,d","ld      l,e",
     "ld      l,h","ld      l,l","ld      l,(hl)","ld      l,a"
3450 DATA "ld      (hl),b","ld      (hl),c","ld      (hl),d","ld      (hl),e",
     "ld      (hl),h","ld      (hl),l","halt    ","ld      (hl),a"
3460 DATA "ld      a,b","ld      a,c","ld      a,d","ld      a,e",
     "ld      a,h","ld      a,l","ld      a,(hl)","ld      a,a"
3470 DATA "add     a,b","add     a,c","add     a,d","add     a,e",
     "add     a,h","add     a,l","add     a,(hl)","add     a,a"
3480 DATA "","adc     a,c","adc     a,d","adc     a,e","adc     a,h",
     "adc     a,l","adc     a,(hl)","adc     a,a"
3490 DATA "sub     a,b","sub     a,c","sub     a,d","sub     a,e",
     "sub     a,h","sub     a,l","sub     a,(hl)","sub     a,a"
3500 DATA "sbc     a,b","sbc     a,c","sbc     a,d","sbc     a,e",
     "sbc     a,h","sbc     a,l","sbc     a,(hl)","sbc     a,a"
3510 DATA "and     a,b","and     a,c","and     a,d","and     a,e",
     "and     a,h","and     a,l","and     a,(hl)","and     a,a"
3520 DATA "xor     a,b","xor     a,c","xor     a,d","xor     a,e",
     "xor     a,h","xor     a,l","xor     a,(hl)","xor     a,a"
3530 DATA "or      a,b","or      a,c","or      a,d","or      a,e",
     "or      a,h","or      a,l","or      a,(hl)","or      a,a"
3540 DATA "cp      a,b","cp      a,c","cp      a,d","cp      a,e",
     "cp      a,h","cp      a,l","cp      a,(hl)","cp      a,a"
3550 DATA "ret     nz","pop     bc","jp      nz,*^","jp      *^",
     "call    nz,*^","push    bc","add     a,#^","rst     0"
3560 DATA "ret     z","ret     ","jp      z,*^","?","call    z,*^",
     "call    *^","adc     a,#^","rst     1"
```

```
3570 DATA "ret    nc","pop    de","jp     nc,*^","out    (^),a",
     "call   nc,*^","push   de","sub    a,#^","rst    2"
3580 DATA "ret    c","exx    ","jp     c,*^","in     a,(^)","call   c,*^",
     "?","sbc    a,#^","rst    3"
3590 DATA "ret    po","pop    hl","jp     po,*^","ex     (sp),hl",
     "call   po,*^","push   hl","and    a,#^","rst    4"
3600 DATA "ret    pe","jp     (hl)","jp     pe,*^","ex     de,hl",
     "call   pe,*^","?","xor    a,#^","rst    5"
3610 DATA "ret    p","pop    af","jp     p,*^","di     ","call   p,*^",
     "push   af","or     a,#^","rst    6"
3620 DATA "ret    m","ld     sp,hl","jp     m,*^","ei     ","call   m,*^",
     "?","cp     a,#^","rst    7"
3630 DATA "","","","","","","",""
3640 DATA "","","","","","","",""
3650 DATA "","","","","","","",""
3660 DATA "","","","","","","",""
3670 DATA "","","","","","","",""
3680 DATA "","","","","","","",""
3690 DATA "","","","","","","",""
3700 DATA "","","","","","","",""
3710 DATA "in     b,(c)","out    (c),b","sbc    hl,bc","ld     *^,bc",
     "neg    a","retn   ","im     0","ld     i,a"
3720 DATA "in     c,(c)","out    (c),c","adc    hl,bc","ld     bc,*^","",
     "reti   ","","ld     r,a"
3730 DATA "in     d,(c)","out    (c),d","sbc    hl,de","ld     *^,de","",
     "","im     1","ld     a,i"
3740 DATA "in     e,(c)","out    (c),e","adc    hl,de","ld     de,*^","",
     "","im     2","ld     a,r"
3750 DATA "in     h,(c)","out    (c),h","sbc    hl,hl","ld     *^,hl","",
     "","","rld    a,(hl)"
3760 DATA "in     l,(c)","out    (c),l","adc    hl,hl","ld     hl,*^","",
     "","","rld    a,(hl)"
3770 DATA "in     f,(c)","","sbc    hl,sp","ld     *^,sp","","","",""
3780 DATA "in     a,(c)","out    (c),a","adc    hl,sp","ld     sp,*^","",
     "","",""
3790 DATA "","","","","","","",""
3800 DATA "","","","","","","",""
3810 DATA "","","","","","","",""
3820 DATA "","","","","","","",""
```

```
3830 DATA "ldi     (de),(hl)","cpi     a,(hl)","ini     (hl),(c)",
     "outi    (c),(hl)","","","",""
3840 DATA "ldd     (de),(hl)","cpd     a,(hl)","ind     (hl),(c)",
     "outd    (c),(hl)","","","",""
3850 DATA "ldir    (de),(hl)","cpir    a,(hl)","inir    (hl),(c)",
     "otir    (c),(hl)","","","",""
3860 DATA "lddr    (de),(hl)","cpdr    a,(hl)","indr    (hl),(c)",
     "otdr    (c),(hl)","","","",""
3870 DATA "","","","","","","",""
3880 DATA "","","","","","","",""
3890 DATA "","","","","","","",""
3900 DATA "","","","","","","",""
3910 DATA "","","","","","","",""
3920 DATA "","","","","","","",""
3930 DATA "","","","","","","",""
3940 DATA "","","","","","","",""
3950 DATA "","","","","","","",""
3960 DATA "","add     ix,bc","","","","","",""
3970 DATA "","","","","","","",""
3980 DATA "","add     ix,de","","","","","",""
3990 DATA "","ld      ix,#*^","ld      *^,ix","inc     ix","","","",""
4000 DATA "","add     ix,ix","ld      ix,*^","dec     ix","","","",""
4010 DATA "","","","","inc     (ix+^)","dec     (ix+^)","ld      (ix+*),#^",""
4020 DATA "","add     ix,sp","","","","","",""
4030 DATA "","","","","","","ld      b,(ix+^)",""
4040 DATA "","","","","","","ld      c,(ix+^)",""
4050 DATA "","","","","","","ld      d,(ix+^)",""
4060 DATA "","","","","","","ld      e,(ix+^)",""
4070 DATA "","","","","","","ld      h,(ix+^)",""
4080 DATA "","","","","","","ld      l,(ix+^)",""
4090 DATA "ld      (ix+^),b","ld      (ix+^),c","ld      (ix+^),d",
     "ld      (ix+^),e","ld      (ix+^),h","ld      (ix+^),l","",
     "ld      (ix+^),a"
4100 DATA "","","","","","","ld      a,(ix+^)",""
4110 DATA "","","","","","","add     a,(ix+^)",""
4120 DATA "","","","","","","adc     a,(ix+^)",""
4130 DATA "","","","","","","sub     a,(ix+^)",""
4140 DATA "","","","","","","",""
4150 DATA "","","","","","","and     a,(ix+^)",""
```

```
4160 DATA "","","","","","","xor     a,(ix+^)",""
4170 DATA "","","","","","","or      a,(ix+^)",""
4180 DATA "","","","","","","cp      a,(ix+^)",""
4190 DATA "","","","","","","",""
4200 DATA "","","","!!!     spezialbefehl mit (ix+*)","","","",""
4210 DATA "","","","","","","",""
4220 DATA "","","","","","","",""
4230 DATA "","pop     ix","","ex      (sp),ix","","push    ix","",""
4240 DATA "","jmp     (ix)","","","","","",""
4250 DATA "","","","","","","",""
4260 DATA "","ld      sp,ix","","","","","",""
4270 DATA "","","","","","","",""
4280 DATA "","add     iy,bc","","","","","",""
4290 DATA "","","","","","","",""
4300 DATA "","add     iy,de","","","","","",""
4310 DATA "","ld      iy,#*^","ld      *^,iy","inc     iy","","","",""
4320 DATA "","add     iy,iy","ld      iy,*^","dec     iy","","","",""
4330 DATA "","","","","inc     (iy+^)","dec     (iy+^)","ld      (iy+*),#^",""
4340 DATA "","add     iy,sp","","","","","",""
4350 DATA "","","","","","","ld      b,(iy+^)",""
4360 DATA "","","","","","","ld      c,(iy+^)",""
4370 DATA "","","","","","","ld      d,(iy+^)",""
4380 DATA "","","","","","","ld      e,(iy+^)",""
4390 DATA "","","","","","","ld      h,(iy+^)",""
4400 DATA "","","","","","","ld      l,(iy+^)",""
4410 DATA "ld      (iy+^),b","ld      (iy+^),c","ld      (iy+^),d",
     "ld      (iy+^),e","ld      (iy+^),h","ld      (iy+^),l","",
     "ld      (iy+^),a"
4420 DATA "","","","","","","ld      a,(iy+^)",""
4430 DATA "","","","","","","add     a,(iy+^)",""
4440 DATA "","","","","","","adc     a,(iy+^)",""
4450 DATA "","","","","","","sub     a,(iy+^)",""
4460 DATA "","","","","","","sbc     a,(iy+^)",""
4470 DATA "","","","","","","and     a,(iy+^)",""
4480 DATA "","","","","","","xor     a,(iy+^)",""
4490 DATA "","","","","","","or      a,(iy+^)",""
4500 DATA "","","","","","","cp      a,(iy+^)",""
4510 DATA "","","","","","","",""
4520 DATA "","","","!!!     spezialbefehl mit (iy+*)","","","",""
```

```
4530 DATA "","","","","","","",""
4540 DATA "","","","","","","",""
4550 DATA "","pop     iy","","ex      (sp),iy","","push    iy","",""
4560 DATA "","jmp     (iy)","","","","","",""
4570 DATA "","","","","","","",""
4580 DATA "","ld      sp,iy","","","","","",""
4590 DATA "rlc     b","rlc     c","rlc     d","rlc     e","rlc     h",
     "rlc     l","rlc     (hl)","rlc     a"
4600 DATA "rrc     b","rrc     c","rrc     d","rrc     e","rrc     h",
     "rrc     l","rrc     (hl)","rrc     a"
4610 DATA "rl      b","rl      c","rl      d","rl      e","rl      h",
     "rl      l","rl      (hl)","rl      a"
4620 DATA "rr      b","rr      c","rr      d","rr      e","rr      h",
     "rr      l","rr      (hl)","rr      a"
4630 DATA "sla     b","sla     c","sla     d","sla     e","sla     h",
     "sla     l","sla     (hl)","sla     a"
4640 DATA "sra     b","sra     c","sra     d","sra     e","sra     h",
     "sra     l","sra     (hl)","sra     a"
4650 DATA "","","","","","","",""
4660 DATA "srl     b","srl     c","srl     d","srl     e","srl     h",
     "srl     l","srl     (hl)","srl     a"
4670 DATA "bit     0,b","bit     0,c","bit     0,d","bit     0,e",
     "bit     0,h","bit     0,l","bit     0,(hl)","bit     0,a"
4680 DATA "bit     1,b","bit     1,c","bit     1,d","bit     1,e",
     "bit     1,h","bit     1,l","bit     1,(hl)","bit     1,a"
4690 DATA "bit     2,b","bit     2,c","bit     2,d","bit     2,e",
     "bit     2,h","bit     2,l","bit     2,(hl)","bit     2,a"
4700 DATA "bit     3,b","bit     3,c","bit     3,d","bit     3,e",
     "bit     3,h","bit     3,l","bit     3,(hl)","bit     3,a"
4710 DATA "bit     4,b","bit     4,c","bit     4,d","bit     4,e",
     "bit     4,h","bit     4,l","bit     4,(hl)","bit     4,a"
4720 DATA "bit     5,b","bit     5,c","bit     5,d","bit     5,e",
     "bit     5,h","bit     5,l","bit     5,(hl)","bit     5,a"
4730 DATA "bit     6,b","bit     6,c","bit     6,d","bit     6,e",
     "bit     6,h","bit     6,l","bit     6,(hl)","bit     6,a"
4740 DATA "bit     7,b","bit     7,c","bit     7,d","bit     7,e",
     "bit     7,h","bit     7,l","bit     7,(hl)","bit     7,a"
4750 DATA "res     0,b","res     0,c","res     0,d","res     0,e",
     "res     0,h","res     0,l","res     0,(hl)","res     0,a"
```

```
4760 DATA "res     1,b","res     1,c","res     1,d","res     1,e",
     "res     1,h","res     1,l","res     1,(hl)","res     1,a"
4770 DATA "res     2,b","res     2,c","res     2,d","res     2,e",
     "res     2,h","res     2,l","res     2,(hl)","res     2,a"
4780 DATA "res     3,b","res     3,c","res     3,d","res     3,e",
     "res     3,h","res     3,l","res     3,(hl)","res     3,a"
4790 DATA "res     4,b","res     4,c","res     4,d","res     4,e",
     "res     4,h","res     4,l","res     4,(hl)","res     4,a"
4800 DATA "res     5,b","res     5,c","res     5,d","res     5,e",
     "res     5,h","res     5,l","res     5,(hl)","res     5,a"
4810 DATA "res     6,b","res     6,c","res     6,d","res     6,e",
     "res     6,h","res     6,l","res     6,(hl)","res     6,a"
4820 DATA "res     7,b","res     7,c","res     7,d","res     7,e",
     "res     7,h","res     7,l","res     7,(hl)","res     7,a"
4830 DATA "set     0,b","set     0,c","set     0,d","set     0,e",
     "set     0,h","set     0,l","set     0,(hl)","set     0,a"
4840 DATA "set     1,b","set     1,c","set     1,d","set     1,e",
     "set     1,h","set     1,l","set     1,(hl)","set     1,a"
4850 DATA "set     2,b","set     2,c","set     2,d","set     2,e",
     "set     2,h","set     2,l","set     2,(hl)","set     2,a"
4860 DATA "set     3,b","set     3,c","set     3,d","set     3,e",
     "set     3,h","set     3,l","set     3,(hl)","set     3,a"
4870 DATA "set     4,b","set     4,c","set     4,d","set     4,e",
     "set     4,h","set     4,l","set     4,(hl)","set     4,a"
4880 DATA "set     5,b","set     5,c","set     5,d","set     5,e",
     "set     5,h","set     5,l","set     5,(hl)","set     5,a"
4890 DATA "set     6,b","set     6,c","set     6,d","set     6,e",
     "set     6,h","set     6,l","set     6,(hl)","set     6,a"
4900 DATA "set     7,b","set     7,c","set     7,d","set     7,e",
     "set     7,h","set     7,l","set     7,(hl)","set     7,a"
```

Profi-Painter CPC

PROFI PAINTER, ein sensationelles Programm zum Malen, Entwerfen und Zeichnen auf CPC Computern. Den berühmten Vorbildern der 32-bit Welt steht PROFI PAINTER kaum nach und übertrifft diese sogar in manchen Punkten. Im einzelnen:

1. Allgemeine Zeichenfunktionen
DAS LINEAL zeichnet Geraden – DAS RECHTECK stellt Rechtecke dar (lassen sich mit Muster oder Farbe ausfüllen) – DAS AUSGEFÜLLTE RECHTECK MIT RUNDEN ECKEN – DER KREIS zeichnet auch Ellipsen (können mit Muster oder Farbe ausgefüllt werden) – DAS POLYGON ist auch ausfüllbar – DER FARBEIMER füllt beliebige Flächen mit Muster oder Farbe – DAS RADIERGUMMI.
Die aktuelle Strichstärke der Funktionen ist jederzeit über ein Menue veränderbar. Es stehen vier Strichstärken (Dicke 0–3) zur Verfügung.

2. Freihand-Zeichenfunktionen
DER PINSEL eignet sich zum Malen auf dem Bildschirm. Verschiedene Pinselformen stehen diesbezüglich zur Verfügung. Mit dem BLEISTIFT kann der Benutzer aus „freier Hand" Linien auf dem Bildschirm zeichnen. DIE SPRÜHDOSE ermöglicht das Sprühen einer Farbe oder eines Musters auf dem Bildschirm. DAS LASSO ermöglicht das Einrahmen und Verschieben beliebiger Bildausschnitte.

3. Sonstige weitere Funktionen
24 zweifarbige Muster stehen zur Verfügung. Eigene Muster können mit dem Mustereditor entworfen werden. So lassen sich alle 24 Muster umdefinieren. Diese werden mit dem betreffenden Dokument abgespeichert, welches ca. eine DIN A 4 Seite umfaßt. Das Dokument läßt sich in alle Richtungen verschieben. Mit dem Auswahlviereck können beliebige Bildstellen zur weiteren Bearbeitung markiert werden.
Weitere Bearbeitungsschritte sind: – verschieben – ausschneiden – in die Zwischenablage kopieren – löschen – ausfüllen – invertieren – Konturen ziehen – horizontal drehen – vertikal drehen – rotieren.

4. Hilfen
Das gesamte Dokument kann mit einem imaginären Raster unterlegt werden. Im Vergrößerungsmodus ist es möglich, jeden Punkt einzeln zu manipulieren. Die Farben können jederzeit geändert werden.
Dokumente können als Hardcopy auf Schneider- und EPSON-Druckern ausgegeben werden.

5. Texteditor
Texte lassen sich problemlos in die Grafik integrieren. Dazu stehen mehrere Zeichensätze in jeweils drei Größen zur Verfügung. Jeder läßt sich zusätzlich fett, kursiv, unterstrichen oder Outline darstellen.

6. Steuerung und Windowing
Nahezu alle Funktionen lassen sich mit Joystick steuern: nur auf das entsprechende Symbol oder Window zeigen und den Feuerknopf betätigen.

Das professionelle deutsche Spitzenprogramm, komplett mit ausführlichem Handbuch, für CPC 464, 664 oder 6128.

PROFI PAINTER CPC DM 198,–*
Lieferbar ab ca. November.

Budget-Manager

Der Budget-Manager ist die universelle Buchführung sowohl für private Zwecke als auch zur Planung, Überwachung und Abwicklung von Budgets jeglicher Art.
In Stichworten:
430 Budgetsätze – 335 Kontensätze – Budget- und Kontenpläne sind per Programm zu erstellen – volle Menuesteuerung erleichtert die Arbeit – Einzelanzeige von Konten, Geldfälligkeiten, Amortisation, Zinsen, Tilgung bis zur kleinen Privatbilanz – Tabellen- und Graphikausgabe auf dem Bildschirm und als Hardcopy auf dem Drucker möglich – zweites Laufwerk wird unterstützt. Für CPC 464 und 664.
BUDGET-MANAGER DM 148,–*

Profimat CPC

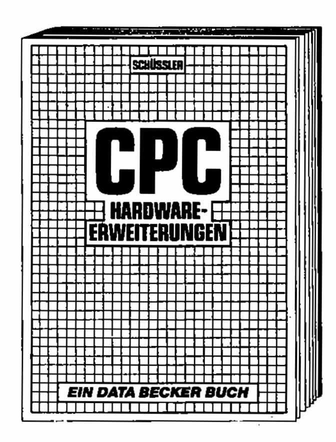
Zur Programmierung in Maschinensprache benötigt man einen Assembler. Doch Assembler ist nicht gleich Assembler. Deshalb gibt es PROFIMAT nun auch für die SCHNEIDER-Rechner. Durch den integrierten Editor wird das Arbeiten mit PROFIMAT zum Vergnügen. Verketten von Quelltexten für besonders lange Assemblerprogramme ist selbstverständlich möglich. PROFIMAT für den SCHNEIDER ist aber mehr als nur ein Assembler, er ist gleichzeitig auch Monitor.
Der absolute Clou dieses Assemblers ist die Möglichkeit, die frisch assemblierten Programme im TRACE-Modus (Einzelschritt-) laufen zu lassen und so jede Änderung an den CPU-Registern verfolgen zu können. PROFIMAT ist frei verschiebbar und kann somit nie in Konflikt mit Ihren eigenen Maschinenprogrammen kommen. Einfache Handhabung durch den komfortablen Editor auch für Anfänger garantiert. Selbstverständlich „beherrscht" der Assembler auch die sogenannten Pseudo-Ops, die bedingtes Assemblieren möglich machen.
PROFIMAT CPC DM 99,–*
Lieferbar ab ca. November für SCHNEIDER CPC 464, 664 und 6128

Mathemat CPC

MATHEMAT, das unentbehrliche Hilfsmittel für Schule, Beruf und Studium, ist nun auch für die SCHNEIDER-Rechner erhältlich.
MATHEMAT beschäftigt sich mit der Geometrie und Algebra sowie mit der Kurvendiskussion. Mit MATHEMAT können Sie beliebige Funktionen ableiten, integrieren und zeichnen lassen. Die Grafiken werden mit einer Skalierung versehen, so daß die Ausdrucke auch in Schule und Studium verwendet werden können. MATHEMAT optimiert selbständig die errechneten Ableitungen, löst Klammern auf etc.
Weitere Programmteile sind der Taschenrechner und der Teil Geometrie/Algebra, in dem Sie Flächen- und Körperberechnungen durchführen können. Eine Hardcopy vom Bildschirm ist jederzeit möglich. Jeder an den SCHNEIDER CPC anschließbare Drucker kann einfach angepaßt werden.
MATHEMAT CPC DM 99,–*
Lieferbar ab ca. November für SCHNEIDER CPC 464, 664 und 6128

* unverbindliche Preisempfehlung
Alle Programme auf 3" Disketten


SCHIEB · WEILER
DAS
CP/M
TRAININGSBUCH
ZUM
CPC
BIOS
TPA
STAT
ED
BDOS
CCP
EIN DATA BECKER BUCH


SCHÜSSLER
CPC
HARDWARE-
ERWEITERUNGEN
EIN DATA BECKER BUCH


Hausbacher
DAS
PROZESSOR
BUCH
ZUM
Z80
Technik & Programmierung
EIN DATA BECKER BUCH